A study and comparison of feature matching

An der Fakultät für Informatik und Mathematik
der Hochschule München
im Studiengang Informatik
bei Prof. Dr. Alfred Nischwitz

eingereichte Masterarbeit von

Herrn Moritz Riegler
geb. 20.02.1987
Matrikelnummer 24440913

wohnhaft in
Augustenstraße 109
80798 München
Email: moritz.riegler@web.de
Tel.: +49 (0) 163 7308147

Kooperation:	Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR) Institut für Methodik der Fernerkundung Oberpfaffenhofen | Münchener Straße 20 82234 Weßling
Betreuung:	Herr Shiyong Cui
Zweitkorrektor:	Prof. Dr.-Ing. Peter Reinartz
Beginn:	13.05.2015
Abgabe:	13.11.2015

Die folgende Erklärung ist in jedes Exemplar der Masterarbeit einzubinden und jeweils persönlich zu unterschreiben.

Riegler.Moritz..... .....20.02.1987...
(Familienname, Vorname) (Geburtsdatum)

München.13.11.2015.. .....WS..... / ...2015.....
(Ort, Datum) (Studiengruppe / WS/SS)

Erklärung
Gemäß § 40 Abs. 1 i. V. m. § 31 Abs. 7 RaPO

Hiermit erkläre ich, dass ich die Masterarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

.....
(Unterschrift)

Danksagung

Zunächst möchte ich Prof. Dr. Alfred Nischwitzdanken, der mich während der gesamten Arbeit mit konstruktiver Kritik und hilfreichen Beiträgen unterstützt hat.
Des Weiteren möchte ich Prof. Dr.-Ing. Peter Reinartz ebenfalls für die Zweitkorrektur der Arbeit danken.
Bei Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), vertreten durch Herr Shiyong Cuimöchte ich ebenfalls bedanken. Die Unterstüzung hat diese Arbeit erst ermöglicht.
Abschließend möchte ich noch meinen Eltern danken, die mir durch ihre finanzielle Unterstützung dieses Studium ermöglicht haben.

Abstract

Die vorliegende Masterarbeit befasst sich mit Analyse und dem Vergleich mehrere Detektoren und Deskriptoren zur Merkmalswiederekennung. Sie wurde vom Autor in Zusammenarbeit und am Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)erstellt. Es werden die theoretischen Grundlagen zu den verschiedenen Merkmalserkennungs- und Merkmalbeschreibungsalgorithmen erläutert. Nachdem der Stand der Technik mit Hilfe ähnliche Arbeiten eruiert wurde, ist ein Quellcode angefertigt worden. Es wurden unterschiedliche Beispieldaten ausgewählt um mit der programmierten Software aussagekräftige Ergebnissdaten zu erzeugen. Diese wurden ausgewertet um eine Aussage darüber treffen zu können, welcher Detektor und Deskriptor in einer gegeben Situation am besten zur Verarbeitung vorgegebener Daten ist.

This master thesis concerns a study and comparison of feature matching. It analyses and compares a few different feature detectors and descriptors. The writer produced it in cooperation with the Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR). The theoretical basics of the different feature detectors and descriptors are elucidated. After a short view at the state of the art by consulting some related work, a source code for calculating the comparison was implemented. Different example data was chosen for significant software results. These were evaluated to find the best detector and descriptor to analyse preexisting images in a given situation.

Inhaltsverzeichnis

Abkürzungen

BRIEF	Binary Robust Independent Elementary Features
BRISK	binary robust invariant scalable keypoints
CenSurE	Center Surrounded Extrema
FAST	Features from Accelerated Segment Test
FREAK	Fast Retina Keypoint
GFTT	Good Features to Track
MSER	maximally stable extremal regions
OpenCV	Open Source Computer Vision Library
ORB	Oriented FAST and Rotated BRIEF
SIFT	Scale Invariant Feature Transform
SURF	Speeded up robust features

1. Einführung

Die vorliegende Masterarbeit befasst sich mit dem Vergleich verschiedener Feature Detektoren und Deskriptoren. Feature kann in diesem Zusammenhang am besten als Merkmale oder markante Punkte übersetzt werden. Sie sind im Idealfall innerhalb einer gewissen Umgebung möglichst einzigartig. Der Prozess der Merkmalsvergleichs (Feature Matching) baut sich aus drei Einzelschritten auf: Finden der Merkmale (Detektion), Berechnung eines Merkmalsvektors zu jedem Merkmal (Deskription) und Vergleichen dieser Vektoren (Matching). Der Fokus dieser Arbeit liegt auf den ersten beiden Schritten. Der letzte wurde immer mit dem gleichen Matchingalgorithmus durchgeführt um einen möglichs objektiven Vergleich zuerzielen.

Vergleichskriterien sind Genauigkeit und Geschwindigkeit, abhängig von einer synthetischen Veränderung der Bilder (wie beispielsweise Rotation oder Skalierung).

Der Vorgang des Feature Matching wurde zu Beginn zur Objekterkennung entwickelt, wird mittlerweile aber auch in anderen Bereichen der Bildverarbeitung angewandt. Z.B. dem Verfolgen von Objekten in Videos oder zur Erkennung bestimmter Bewegungen im Sichtfeld eines Roboters. Ein weiterer Anwendungsbereich ist das Stiching, bei dem aus verschiedenen Einzelbildern, die sich teileweise überschneiden, ein größeres zusammenhängendes Bild erstellt wird. Vorzeigebeispiele sind Panoramabilder oder Bilder der Erde, welche aus vielen Satellitenbildern zusammengefügt werden.

1.1. Beschreibung der Aufgabenstellung

In dem Vorschlag für diese Arbeit wurden als Schwerpunkte das Einlesen in das Thema anhand schon existierender Veröffentlichungen (siehe dazu 3), das Erabeiten eine Software (5), welche als Ergebnisse Werte zum numerischen Vergleich der verschiedenen Detektor-Deskriptor-Kombinationen ausgibt. Final sollte eine Arbeit mit dem Fokus auf den Resultaten der Software verfasst werden.

In der Aufgabenstellung wurden ebenfalls schon einige der üblichen Detektoren und Deskriptoren genannt. Es sollten dazu verschiedene Bilddaten verwendet werden und besonders Genauigkeit und Geschwindigkeit der Algorithmen untersucht werden.

Zur Veränderung der Bilder wurden am Anfang noch keine genauen Operationen genannt. Letzlich wurden die Algorithmen auf künstlich veränderte Bilder losgelassen, die mit Hilfe von Rotationen, Skalierungen, Beleuchtungsveränderungen und Glättungen bearbeitet wurden.

Die Verwendung einer bestimmten Software wurde nicht vorgegeben, da die Software keine Intaktion mit anderen Programmen benötigen würde (ergo Stand-Alone agieren). Gewählt wurde C++ mit der Open Source Library OpenCV. Als Programmierumgebung wurde der Qt Creator verwendet.

1.2. Lösungskonzept

Da in der Aufgabenstellung relativ viele Freiheiten eingeräumt wurden, musste zuerst ein grobes Lösungskonzept erstellt werden um den Vergleich der Detektoren und Deskriptoren umfassend herauszuarbeiten.

Die Erstellung dieser Arbeit wurde in folgende Schritte gegliedert:

1. Recherche der verschiedenen Algorithmen und vorhergehender Arbeiten zum Thema
2. Auswahl einer passenden Softwareumgebung
3. Auswahl von Testbildern
4. Erstellen der Software
5. Interpretieren der Ergebisse

Diese Schritte sind in der vorliegenden Masterarbeit in den folgenden Kapiteln erläutert. Welche Bereiche, an welcher Stelle der Arbeit zu finden sind wird im folgenden Unterkapitel (1.3) angeführt.

Die entscheidenden Ideen sind letztlich das mehr Kombinationen der verschiedenen Detektoren und Deskriptoren verglichen wurde als in anderen Arbeiten. Dazu ist die Software so erstellt worden, dass es einfach ist auch andere Bilder, Bildserien, Detektoren, Deskriptoren aber auch Modifikationen der Bilder zu verwenden. Während es einfach ist ein geglättetes Bild mit dem Ursprungsbild zu vergleichen, weil eben die Positionen der Merkmale bei einer Glättungen an der gleichen Stelle bleiben, müssen bei z.B. einer Rotation auch die neuen Koordinaten der markanten Punkte errechnet werden. Hier wurde ein passende Transformation mit implementiert. Auch wurden Bilder anhand sovieler unterschielicher Kriterien wie in keiner der anderen Vorarbeiten.

Der letzte Schritt ist das Anfertigen der vorliegenden Arbeit, die einen guten und übersichtlichen Vergleich der verschiedenen Algorithmen ermöglicht. In der Arbeit sollten auch die Ergebnisse aufbereitet werden und ein Ausblick gemacht werden.

1.3. Aufbau der Arbeit

In der Einführung (1) wird das Problem, dessen Lösung in der Arbeit dokumentiert ist, genau beschrieben.

Das zweite Kapitel (2) beschäftigt sich mit den theoretischen Grundlagen zur Merkmalswiederekennung. Hier werden die verschiedenen Detektoren und Deskriptoren kurz beleuchtet.

Im dritten Kapitel (3) wird der Stand der Technik anhand ähnlicher Arbeiten eruiert. Hier liegt der Fokus auf den Ergebissen von Abhandlungen zu vergleichbaren Themen.

Das Kapitel Daten und Ergebisse (4) beschreibt die Daten und die entsprechenden Ergeb-

nisse, die im Laufe der Arbeit erreicht wurden.

Der Aufbau des Programmcodes wird im vierten Kapitel (5) ausformuliert. An dieser Stelle wird die Implementierung, welche eine Eigenleistung ist, erläutert.

In der Konklusion (6) werden die Ergebisse noch mal zusammengefasst und kurz mit den vorhergehenden Arbeiten verglichen.

Der Ausblick (7) schließt als letztes Kapitel diese Arbeit ab und bietet dazu noch eine Sicht auf (eventuelle) Möglichkeiten der Analysefortführung.

2. Theoretische Grundlagen

Auch wenn dieses Kapitel ein entscheidender Teil ist wird nur kurz auf die verschiedenen Algorithmen eingegangen, da deren Funktionsweise bereits an vielen Stellen, insbesondere in den Quellen des aktuellen Kapitels, nachgelesen werden kann.

Wie in der Einführung schon geschrieben besteht der Vorgang des Merkmalsvergleichs (Feature Matching) aus drei Einzelschritten: Merkmalsextraktion (Detektion), Berechnung eines Deskriptorvektors zu jedem Merkmal (Deskription) und Vergleichen der Deskriptoren (Matching).

Da im Fokus dieser Arbeit die ersten beiden Schritte liegt, wird auf sie und deren Werkzeuge deutlich ausführlicher als auf den Dritten eingegangen.

2.1. Merkmalsextraktion

Bevor auf die Merkmalsextraktion an sich eingangen wird, folgt noch eine kurze Beschreibung welche Punkte möglichst gut als Merkmal geeigent sind. Die folgenden Absätze und Bilder sind [1] entnommen.

Die am besten wieder zuerkennenden Merkmale sind verständlicherweise die Ecken von Objekten, wie in dem folgenden Bild zu sehen ist:

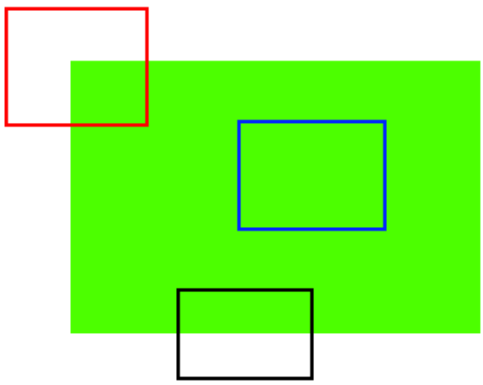


Abbildung 2.1.: Einfache Merkmale

Am besten eindeutig wiederzufinden ist der rote umrahmte Bereich, weil er eben einzigartig ist (solange das Bild nicht rotiert wird). Den blau oder gar schwarz gekennzeichneten Bereich als Merkmal zu verwenden hilft sicher nicht weiter.

Hier noch ein praktisches Bild zum besseren Verständniss von Merkmalen:

Abbildung 2.2.: Gute Merkmale

Die Bereiche **A** und **B** sind für die eindeutige Zuordnung von Merkmalen zu einem passenden Punkt in einem anderen Bild nicht zuträglich. Die Bereiche **C** und **D** können die Lokalisierung eines Objektes schon mal auf eine Strecke einschränken, besser ist es aber Merkmale wie **E** und **F** zu Finden und zu Beschreiben, da diese einzigartig sind und somit wieder eindeutig zu geordnet werden können.

Es werden in dieser Arbeit nicht alle existierenden Detektoren und Deskriptoren behandelt, weil das in einem bestimmten zeitlichen Rahmen eben nicht möglich ist. Es wurden die Algorithmen ausgewählt die in OpenCV implementiert sind. Weitere werden aber in dem Kapitel 7 zumindest erwähnt.

2.1.1. SIFT

Als erstes wird hier auf den bekanntesten Merkmalsdetektor eingegangen, mit ihm müssen sich alle Folgenden messen. Er heißt SIFT und wurde 2004 von David G. Lowe veröffentlicht. Quelle für dieses Unterkapitel und dessen Bilder ist [2].

Die Abkürzungen SIFT steht für "Scale-invariant feature transform", zu deutsch etwa skaleninvariante (maßstabsunabhängige) Merkmalstransformation.

Um eine Merkmal zu finden, das ganz besonders skaleninvariant ist wird zuerst eine Gauß-Laplace-Pyramide erstellt. Das Verfahren ist in [3] erklärt. Dadurch werden sozusagen verschiedene Skalenebenen erzeugt. Um nun einen möglichen Merkmalspunkt zu finden wird jeder Pixel mit seinen direkten Nachbarpixeln verglichen. Durch die Gauß-Laplace-Pyramide sind hier auch die Nachbarn in der Skalenrichtung vorhanden. Wie in im folgenden Bild zu sehen hat jeder Pixel 26 Nachbarppixel.

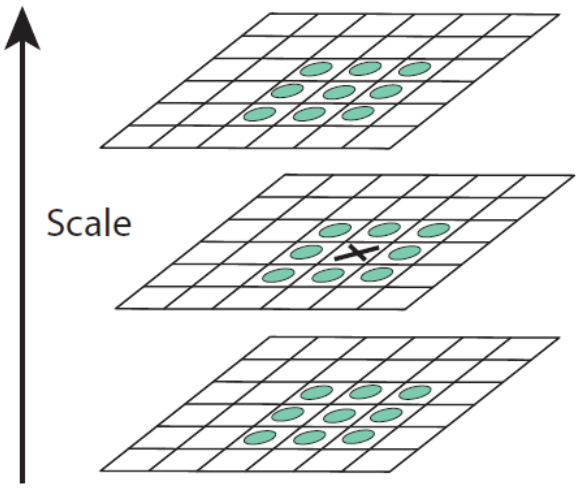


Abbildung 2.3.: Nachbarn der Merkmale

Nur wenn er einen größeren oder kleineren (Grau-)Wert als alle seine Nachbarn besitzt wird er als Merkmal in Betracht gezogen. In der folgenden Abbildung ist ein Bild und die darin gefundenen Merkmale zu sehen.

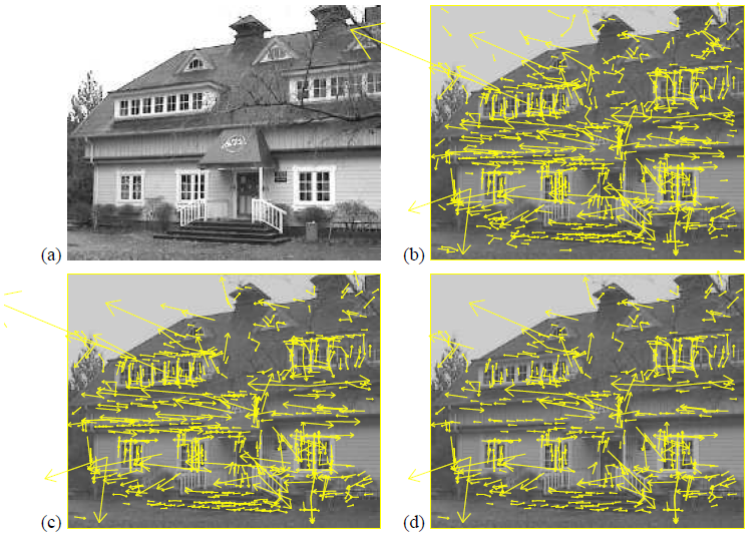


Abbildung 2.4.: Auswahl der Merkmale

In (a) ist das zu untersuchende Graubild (233x189 Pixel) zu sehen. (b) Die initialen 832 Merkmale wurden gefunden (wie oben erklärt). Die Vektoren stellen Maßstab, Orientierung und Ort der Punkte dar. (c) Nach Anwendung eines Schwellwerts (Threshold) sind noch 729 Merkmale übrig. (d) Hier wurden Merkmale an Kanten eliminiert (wie **C** und **D** in 2.2). So werden am Ende möglichst nur Merkmale, die wirkliche Ecken im Bild darstellen weiter verarbeitet.

Bevor der Deskriptor angewandt wird, wird jedem Merkmal noch eine Orientierung zuge-

wiesen, sie wird mit Hilfe eines Orientierungshistogramm bestimmt. Dadurch und durch den aufwendigen Deskriptor soll SIFT rotationsinvarianter als die anderen Algorithmen sein.

Ein Nachteil des SIFT-Algorithmus ist das, wenn er in einer kommerziellen Software verwendet wird, Lizengebühren anfallen.

2.1.2. SURF

Quellen für das folgende Unterkapitel sind [4] und [5]. Zur genaueren Erklärung der Funktionsweise des Algorithmus können diese auch zu Rate gezogen werden.

Bei SURF wird der ein Boxfilter angewandt um Merkmale zu finden. Die zweite Ableitung wird so einfacher approximiert und somit ist der Rechenaufwand geringer als bei SIFT.

Bei SURF ist die Option vorhanden die Rotation von Merkmalen ganz auszuschließen. Damit ist der Algorithmus nicht mehr rotationsinvariant (bei Winkeln über und unter ca. 15 Grad zum Ausgangsbild) dafür aber natürlich schneller.

Da der Fokus bei SURF besonders auf der Geschwindigkeit liegt ist er schneller als SIFT dafür aber nicht so gut (wie SIFT) bei Veränderung des Gesichtspunkts (Viewpoint Change) und Helligkeitsänderungen. Rotation und Unschärfe sollen aber ähnlich gut wie von SIFT gehandelt werden.

2.1.3. FAST

Als nächstes wird hier kurz auf den FAST Detektor eingegangen. Als Quelle (auch für die Bilder) dient ebenfalls ein Paper ([6]).

FAST bedeutet ausgeschrieben: Merkmale aus einem beschleunigtem Kreisabschnittstest (features from accelerated segment test). In diesem Verfahren werden gar keine Ableitungen gebildet sondern ein Pixel einfach mit den Pixeln in seiner Umgebung verglichen, damit auch eine gewisse Größenunabhängigkeit gegeben ist werden meist aber nicht die unmittelbaren Nachbarpixel betrachtet.

Standardmäßig wird (wie auch in der Software der Arbeit) die graue Maske (zu sehen in der folgenden Abbildung), mit 16 Pixeln, verwendet. Um das ganze noch weiter zu beschleunigen werden erst die Pixel mit Nummer 1 und 9 betrachtet, sind sie beide heller oder dunkler als der Pixel in der Mitte, werden noch die Pixel 5 und 13 angesehen. Sollten alle vier heller oder dunkler sein, wird der Wert des Pixels in der Mitte auch noch mit den anderen Pixeln verglichen, sind letzlich 12 der 16 Pixel heller oder dunkler als der betrachtete Pixel, wird er als Merkmal vermerkt.

Im folgenden Bild sind die verschiedenen Masken, welche zur Wahl stehen, zu sehen:

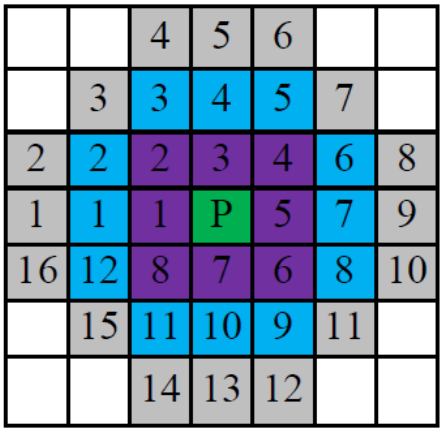


Abbildung 2.5.: Masken für FAST-Algorithmus

Um zu verhindern das benachbarte Pixel als verschiedene Merkmale erkannt werden, obwohl sie eigentlich nur ein Eckpunkt im Bild sind, kann noch eine sogenannte Non-maximal Suppression angewandt werden. Dafür wird eine Auswertungsfunktion (score function) berechnet. Ihr Wert ist die Summe der absoluten Differenzen eines Merkmalpixels zu den anderen 16 Pixeln außen herum. Wenn zwei Merkmale zu nah zusammen liegen, wird der mit dem geringeren Wert der Auswertungsfunktion verworfen.

Die Nachteile des Verfahrenens liegen auf der Hand. Es ist weniger größenunabhängig als die anderen Algorithmen. Dazu hängt der Erfolg auch noch deutlich mehr von der tatsächlichen Bilddatei ab. Eine Orientierung der Merkmale ist gar nicht vorhanden. Zusätzlich ist ein Schwellwert immer fehleranfällig, weil eben als Schwellwert nur der Wert eines einzelnen Pixels (dem in der Mitte) verwendet wird. Durch seine Funktionsweise macht der FAST-Algorithmus seinem Namen aber eben alle Ehre (Er ist benötigt deutlich weniger Zeit um Merkmale zu finden als andere Algorithmen.).

2.1.4. GFTT

Um den GFTT-Algorithmus zu erklären wird er kurz auf den Harris Eckendetektor eingegangen, weil GFTT auf ihm basiert. GFTT ist in [7] ausführlich erläutert. Zum Harris Eckendetektor wird [8] zurate gezogen.

Beim Harris Eckendetektor wird eine Funktion, für die Veränderung der Helligkeit in den beiden Bildrichtungen, definiert. Um eine Ecke zu finden muss die Funktion maximiert werden. Letzlich zeigen die Eigenwerte einer Matrix auf ob es sich um eine Ecke, Kante oder eben eine flache Region handelt.

Zur Veranschaulichung dient folgendes Bild:

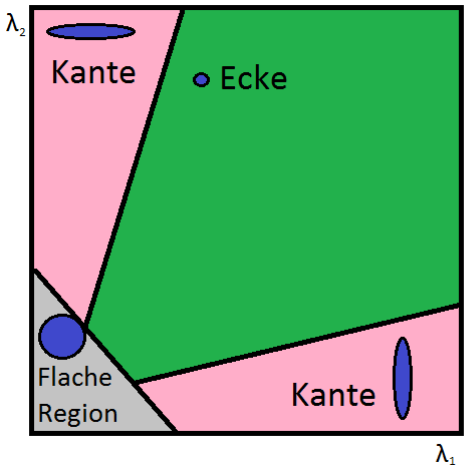


Abbildung 2.6.: Typ des Bereichs nach dem Harris Eckendetektor

Sind λ_1 und λ_2 groß und etwa gleich, ist der betrachtete Punkt eine Ecke und kann somit als Merkmal genutzt werden. Ist $\lambda_1 >> \lambda_2$ oder anders herum handelt es sich um eine Kante. Trifft nichts davon zu ist der Punkt Teil einer flachen Region.

Auswertungsfunktion des Harris Detektors sieht folgendermaßen aus:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

Die Abkürzungen GFTT steht für Good Features to Track, was Merkmale, welche gut zu Tracken (Verfolgen) sind. Merkmalswiederekennung wird oft zum Verfolgen von Objekten in mehreren Kamerabilder benutzt.

Für GFTT wird aber diese Funktion verwendet:

$$R = min(\lambda_1, \lambda_2)$$

Damit können Punkte einfacher klassifiziert werden. Die Bereiche sehen damit folgendermaßen aus:

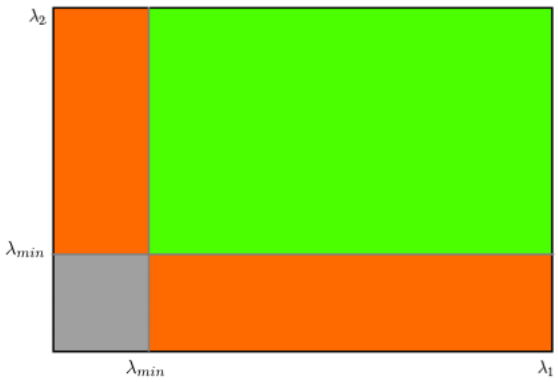


Abbildung 2.7.: Typ des Bereichs nach GFTT

(Quelle für das Bild ist [9]).

Das Verfahren liefert bessere Resultate als der ursprüngliche Harris Eckendetektor und ist dazu noch schneller.

2.1.5. MSER

Das folgende Unterkapitel ist der Quelle [10] entnommen.

Als nächstes wird hier der MSER-Algorithmus beschrieben. Der Name ist eine Abkürzung für maximal stabile Extremwertregionen (Maximal stable extermal regions).

Hier werden nicht nur Punkte sondern ganze Regionen von variabler Größe (über einen Schwellwert) ausgewählt um in einem Folgebild wiedererkannt zu werden. Als Grundlage wird hier die Epipolargeometrie verwendet. Sie liefert eine Aussage darüber wie ein Bild in ein anderes Bild (z.B. Folgebild) überführt werden kann. Dadurch ist die Menge der Merkmale meist geringer als, die anderer Algorithmen.

Weil ganze Regionen verwendet werden ist MSER stabiler gegen über Verformungen oder Verdrehungen des Bildes bzw. der abgebildeten Objekte.

2.1.6. ORB

Die Grundlage für dieses Unterkapitel liefert das Paper [11].

Die Abkürzung ORB steht für Oriented FAST and Rotated BRIEF. Es werden Algorithmen benutzt, die an einer anderen Stelle erklärt werden. Für den Detektor siehe auch das Unterkapitel zu FAST (2.1.3).

Es wird aber das sogenannte oFAST verwendet, das steht für orientiertes FAST. Zusätzlich zum normalen FAST wird zu jedem Merkmal auch noch eine Richtung berechnet. Es wird FAST mit dem Radius neun verwendet. In 2.5 ist FAST-9 violett zu sehen. Bevor ein Merkmal weiterverarbeitet wird, werden aus den schon gefundenen Merkmalen noch welche, die nicht optimale sind, mit Hilfe des Harris Eckendetektors ausgeschlossen. Nun wird den Merkmalen eine Richtung zugewiesen. In der Quelle [11] kann das Verfahren dazu genau nachgelesen werden.

2.1.7. STAR

Dem STAR-Algorithmus liegt CenSurE (Center Surrounded Extrema) zugrunde. Hier wird eine geometrische Forum um das Merkmal herum betrachtet. (Das Merkmal selbst liegt in dessen, Zentrum. Z.B. dem Kreismittelpunkt.)

Als Quellen dienen hier: [12] und [13].

CenSurE verwendet verschiedene geometrische Formen, welche das Merkmal umgeben. Auch wenn ein Kreis am besten wäre, sind Bilder (aufgrund der Zusammensetzung aus Pixeln) eben diskret. CenSurE verwendet deshalb Formen wie Quadrat, Sechs- oder Achtecke. STAR approximiert einen Kreis mit zwei Quadraten wobei eines um 45 Grad gedreht ist.

Es werden (wie auch beim SIFT) auch noch die Nachpixel über und unter (in der Gauß-Laplace-Pyramide) dem Punkt betrachtet. Dann wird eine non-Maxima Supression wie bei FAST verwendet und die Kantenpuntke (die keine Ecken sind) mit Hilfe des Harris Eckendetektors aussortiert.

Jetzt ist der zweite Schrit der Merkmalsextraktion abgeschlossen. Die Merkmale sind damit gefunden und werden jetzt Beschrieben. Das folgende Kapitel erleuchtet diesen Schritt näher.

2.2. Merkmalsbeschreibung

Um ein Merkmal, das von einem Detektor gefunden wurde, in einem Folgebild wiederzufinden muss es möglichst eindeutig beschrieben werden. In diesem Kaptitel werden die verwendeten Deskriptoren (Beschreiber) kurz beleuchtet.

Ein Deskriptor ist ein Vektor, dessen Länge und Einträge von dem Deskriptor-Algorithmus bestimmt werden. Auch wird zu jedem Deskriptor-Algorithmus die Distanz, mit welcher das passende Merkmal im anderen Bild gefunden wird, angegeben. Zum Berechnen der Abstände werden hier der euklidische Abstand und die Hammingdistanz verwendet. Der euklidische Abstand zweier Vektoren wird wie bekannt berechnet:

$$d(x,y) = \|x-y\|_2 = \sqrt{(x_1-y_1)^2+\cdots+(x_n-y_n)^2} = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

Die Hammingdistanz ist einfach die Summe der unterschiedlichen Einträge zweier Vektoren:

$$d(x,y) := |\{j \in \{1,\ldots,n\} \mid x_j \neq y_j\}|$$

Sie lässt sich schneller berechnen, aber es wird ein längerer Vektor benötigt um den ganzen Raum abzudecken. Letztlich ist ein Algorithmus, der die Hammingdistanz verwendet, grundsätzlich schneller, als die Anderen.

2.2.1. SIFT

Grundlage bietet wie schon in 2.1.1 das Paper von David G. Lowe [2]. Es dient auch als Quelle für das Bild.

Am besten lässt sich die Erstellung des SIFT Deskriptorvektor anhand einer Abbilldung erklären:

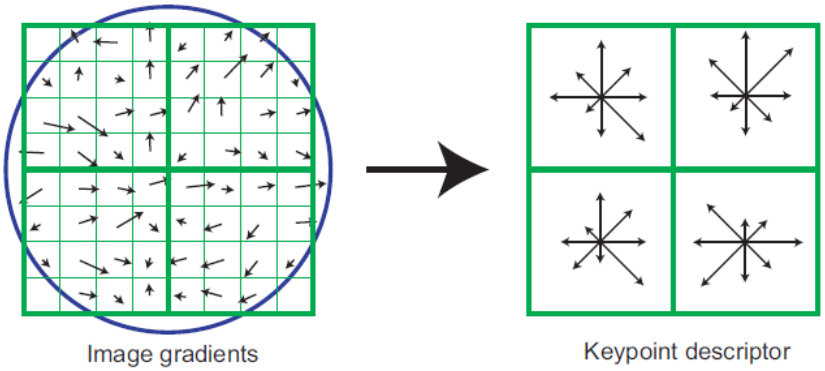


Abbildung 2.8.: Auswahl der Merkmale

Als erstes werden die Länge und Richtung des Gradienten jedes Pixels in der Umgebung des Merkmals berechnet (wie im linken Teil der Abbildung zu sehen). Sie werden noch mit Hilfe einer Gaußglocke gewichtet (Basis der Glocke ist der blaue Kreis, ebenfalls im linken Bild markiert). Nun werden die Gradienten von jeweils 4x4 Pixel zusammengefasst. Es werden die einfach die Komponenten aller Gradienten in acht vorgegeben Richtungen aufsummiert. Im Bild wird eine 2x2 Deskriptormatrix (rechts), welche aus einer 8x8 Region berechnet wurde, gezeigt. Verwendet wird aber eine 4x4 Matrix, welche aus einer Region von 16x16 Pixeln ermittelt wird.

Die Gewichtung mit der Gaußglocke verhindert das sich abrupte Änderungen in der Nähe des Merkmals zu sehr aus den Deskriptorvektor auswirken.

Um von der Martrix auf einen Vektor zu kommen, werden die Einträge einfach (in einer festgelegten Reihenfolge) in einen Vektor übertragen. Der Vektor hat 128 Einträge da für jede Unterregion (16 Stück) acht Zahlenwerte (jeder für eine Richtung) gespeichert werden müssen.

Damit sich Änderungen der Helligkeit nicht auf den Vektor auswirken, wird er am Ende noch normiert.

Um Vektoren dieser Art zu Vergleichen muss der euklidische Abstand verwendet werden.

Sind bei dem Vergleichen der Merkmale am Ende (in Rahmen einer gewissen Toleranz) zwei mögliche Kanidaten (für das Merkmal im anderen Bild) gleich nah, wird das Merkmal nicht zugewiesen. So werden bei unsicheren Punkten Fehler ausgeschlossen.

Durch die recht aufwendige Beschreibung ist SIFT besonders rotationsinvariant.

2.2.2. SURF

Quellen für das folgende Unterkapitel sind [4] und [5].

SURF benutzt zum Beschreiben der Ausrichtung der Merkmale auch einen quadratischen Bereich, der das Merkmal umgibt.

Wie schon in 2.1.2 erwähnt, kann SURF auch ohne die Beschreibung der Orientierung benutzt werden. Das ist selbstverständlich schneller, wird in den Ergebnissen der Arbeit (zum besseren Vergleich mit den anderen Algorithmen) aber außer Acht gelassen.

Um die Ausrichtung eines Merkmals zu ermittelen wird die Haar-Wavelet-Antwort in x- und y-Richtung benutzt. Wie in der folgenden Abbildung zu sehen.

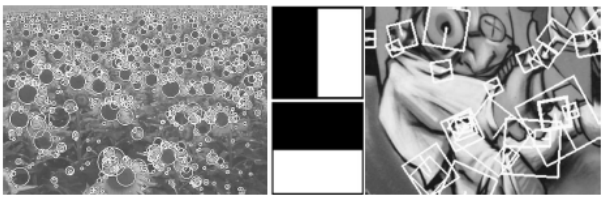


Abbildung 2.9.: Haar-Wavelet-Antwort

Links sind die gefundenen Merkmale in einem Feld zu sehen (hier nicht von Bedeutung).

In der Mitte die Haar-Wavelet-Varianten, die von SURF verwendet werden. Rechts sind die Merkmale in einem Bild mit ihrer Hauptrichtung zu sehen. Die Größe hängt davon ab in welcher Skalenebene das Merkmal gefunden wurde.

Der Rest läuft jetzt recht ähnlich zu den Operationen des SIFT-Algorithmus. Nur das als umgebendes Quadrat eben schon ein, in der Hauptrichtung des Merkmals, orientiertes Quadrat verwendet wird. Dazu werden hier für die Subregionen nicht acht Richtungen (wie bei SIFT) sondern nur passend zu den Haar-Wavelet-Antworten (zwei Richtungen) verwendet.

Als Distanz zum Vergleichen der Vektoren muss wieder der euklidische Abstand verwendet werden. Durch weniger Richtungen sind weniger Rechenoperationen nötig und somit ist SURF schneller als SIFT.

2.2.3. BRIEF

Weitere Lektüre zu diesem Unterkapitel bietet die Quelle [14].

Brief is der erste Binärdeskriptor, welcher hier beschrieben wird. Er erzeugt keinen Deskriptorvektor mit Float-Werten als Einträgen, sondern nur einzelnen Bits (d.h. als Wert gibt es nur 0 und 1). Damit können die Vektoren (mit der oben genannten Hammingdistanz) einfacher und damit schneller verglichen werden.

BRIEF steht für binäre robuste unabhängige grundlegendene Merkmale (binary robust independant elementary features).

Es werden ebenfalls die Pixel in der Umgebung des Merkmals zu seiner Beschreibung herangezogen. Es werden aber eben keine Richtungen ermittelt, sondern einfach (nach einer Glättung) die Grauwerte bestimmter Pixel in der Umgebung miteinander verglichen. Die Länge des Vektors kann theoretisch jede Zahl sein. Es werden praktisch aber immer Werte wie 128, 256 oder 512 benutzt. Es werden aber nicht die direkten Nachbarn des Merkmals betrachtet sondern bestimmte Pixel in der Umgebung, die sich natürlich immer an den gleichen Stellen realitv zum Merkmal befinden müssen.

Als mögliche Schemen zu Auswahl (der betrachteten Pixel) werden in dem Paper verschiedene Bespiele gezeigt:

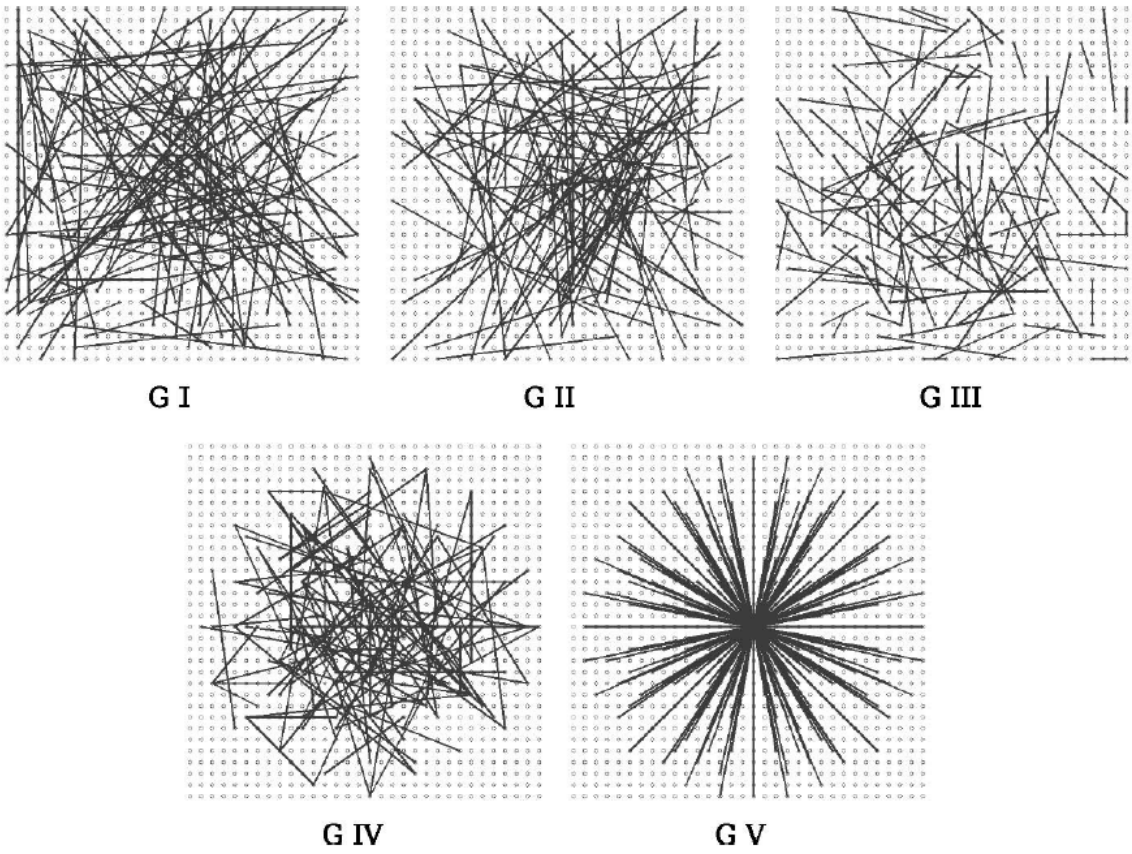


Abbildung 2.10.: Auswahl der betrachteten Pixel(BRIEF)

Jedes Pixelpaar, das mit einer schwarzen Linie makiert ist, bekommt einen Eintrag in dem Merkmalsvektor. Es werden einfach die beiden Grauwerte verglichen, ist der Erste kleiner wird eine 1 weitergegeben, ist er größer eine 0. Welcher der erste Wert ist wird auch nach einem Schema immer gleich gewählt. Jeder einzelne Pixel kann wird aber nur einmal benutzt.

Dadurch gibt BRIEF am Ende einen Merkmalsvektor aus der mit denen der Punkte im zweiten Bild verglichen werden kann. Das das Verfahren recht schnell arbeitet (weil praktisch keine Rechenoperationen vorgenommen werden müssen) dafür aber kaum Rotationen und Größenunterschiede handeln kann, wird später in dieser Arbeit zu sehen sein. Es gibt natürlich auch Methoden wie BRIEF modifiziert werden kann um auch mit derartigen Bildveränderungen zurechtkommen.

2.2.4. BRISK

Quelle für dieses Unterkapitel ist die Veröffentlichung zu BRISK: [15]. Darin ist auch die Funktionsweise des BRISK Merkmalsdetektors erklärt, da er aber in OpenCV (noch) nicht implementiert ist, ist er nicht Teil der Auswertung in dieser Arbeit. (Der Detektor basiert auf FAST. Mehr dazu ist in der Quelle zu finden ([15]).

Der Name BRISK steht für binäre robuste skalierbare Merkmale (binary robust invariant scalable keypointes). Der Unterschied zu BRIEF geht schon aus dem Namen hervor, auch wenn BRISK eben falls einen binären Mekrmalsvektor ausgibt, ist er besser in der Verarbeitung von Folgebildern unterschiedlicher Größe.

BRISK benutzt ähnlich wie SURF als Interessenregion um den Punkte ebenfalls ein Quadrat, das schon (vor den ersten Berechnungen) passend rotiert wird. Das Verfahren selbst ist sehr ähnlich zu BRIEF, es wird aber immer ein festes (und damit das selbe) Schema zu Auswahl der Punkte, die verglichen werden, benutzt.

In der folgenden Abbildung sind die verwendeten Pixel zu sehen:

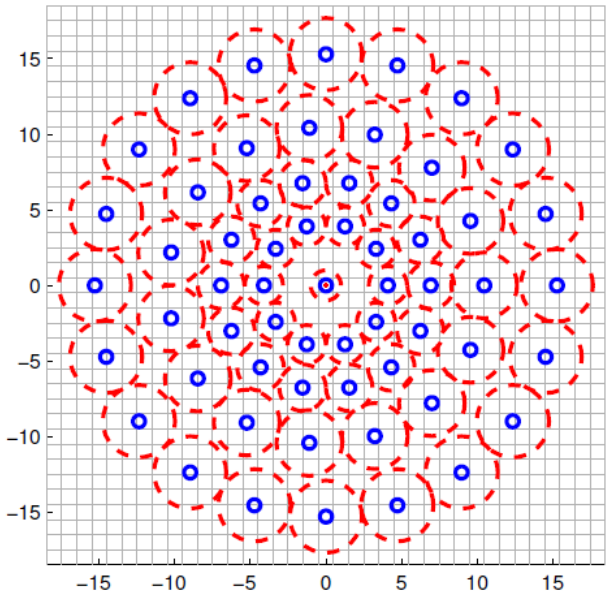


Abbildung 2.11.: Auswahl der betrachteten Pixel(BRISK)

Die blauen Bereiche zeigen an welchen Stellen Pixel herausgegriffen werden. Die roten Kreise zeigen das Schema der Auswahl. Es fließen mehr Punkte, die nah am Merkmal liegen, in den Aufbau des Merkmalsvektors mit ein.

Dazu werden auch weniger Punkte (als bei BRIEF) verwendet, beispielsweise kann ein einzelner Pixel auch in mehrere Einträge des Merkmalsvektors eingehen, da er mit mehreren anderen Pixeln verglichen wird.

Wie bei BRIEF auch wird die Hammingdistanz verwendet um die Merkmale mit einander zu vergleichen. Angeblich ist ist BRISK, bei fast gleicher Genauigkeit, deutlich schneller als SIFT und SURF.

2.2.5. FREAK

Als nächstes wird hier kurz die Funktionsweise des FREAK-Deskriptors erklärt. Als Quelle dient das Paper, in dem der Algorithmus vorgestellt wurde ([16]).

Der Name leitet sich von schnellen Netzhautmerkmalen ab (Fast Retina Keypointes). Wie der Name schon sagt ist der Algorithmus von der menschlichen Netzhaut inspiriert.

Die Erstellung des Merkmalsvektors läuft bei FREAK ähnlich ab wie bei BRISK, die verschiedenen Teilregionen werden aber einzeln geglättet um detaillierte Informationen zu behalten. Die Größe des Gaußkerns wird an den einzelnen Teilbereich angepasst.

Welche Teilbereiche um das Merkmal selber betrachtet werden ist der folgenden Abbildung zu entnehmen:

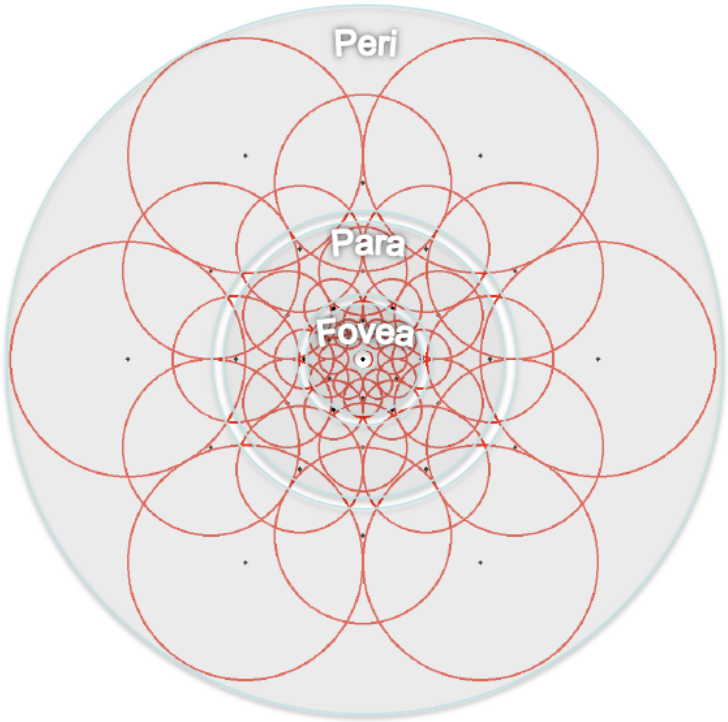


Abbildung 2.12.: Auswahl der betrachteten Teilbereiche

Da sich die Teilbereiche teilweise überlappen ist eine gewisse Redundanz gegeben, sie kostet aber auch Rechenzeit.

Als Punkte dienen wieder die Mittelpunkte der Kreis (wie bei BRISK). Sie werden verglichen und nach einem festen System in den Merkmalsvektor übertragen.

Der Deskritor ist sehr robust gegenüber aller Art von Deformationen, aber dafür langsamer als die anderen Binärdeskriptoren. Es wird ebenfalls die Hammingdistanz verwendet.

2.2.6. ORB

ORB bietet neben einem Detektor auch noch einen Deskriptor, der in diesem Unterkapitel beschrieben wird. Quelle ist wieder eine entsprechende Veröffentlichung: [11].

Da ORB für Oriented FAST (Detektor) und Rotated BRIEF (Deskriptor) steht, wird zum Beschreiben der Merkmale ein Algorithmus verwendet, der BRIEF sehr ähnlich ist. Es bietet sich an, da BRIEF nahezu keinerlei Rotation in der Bildebene handeln kann, zusätzlich noch eine Rotationsüberprüfung durchzuführen. Nachdem die Werte für die Pixel in der Umgebung des Merkmals gespeichert sind, wird dieser Bildbereich (als Matrix) noch 29 mal um 12 Grad rotiert und ebenfalls gespeichert. Da immer um die selben Gradzahlen gedreht wird, kann zu jedem Pixel (abhänigig von seiner Postion) die neue Position (nach der Drehung) einfach in einer Lookup-Tabelle, die nur einmal (für jeden Winkel) angelegt werden muss, nachgeschlagen werden. Damit kosten diese Drehungen kaum Rechenzeit.

Der Matchingprozess dauert natürlich länger als beim reinen BRIEF, weil jeder Merkmalsvektor eigentlich mit 29 mal sovielen Werten vergleichen werden müsste. Mit einer optimierten Suche (z.B. mit Hilfe von k-d-Bäumen) kostet der Matchingprozess nicht wirklich die 29-fache Zeit.

Es wird auch die Hammingdistanz als Abstandmessung verwendet.

2.3. Matching

Nun folgt der dritte und letzte Schritt des Merkmalsvergleichs. Hier werden die Merkmalsvektoren verglichen und damit zu jedem Merkmal ein hoffentlich passender Punkt in dem anderen Bild gefunden.

Unter Matching ist zu verstehen, dass zwei Punkte in verschiedenen Bilder zusammengeführt werden und damit ein und das selbe Merkmal sind. Im Gegensatz zur Detektion und Deskription wurde für die Ergebisse der Software immer der gleiche Matchingalgorithmus benutzt, da eben die anderen beiden Schritte des Merkmalsvergleichs das Hauptthema dieser Arbeit sind.

Es wurde ein sogenanntes "brute force"-Matching benutzt, hier werden auf der Suche nach deinem passenden Merkmal bzw. dessen Deskriptorvektors werden alle möglichen Kanidaten durchgesehen um das passende Merkmal im anderen Bild zu finden.

Das ist natürlich recht langsam, weil keinerlei Optimierung stattfindet, aber hier eben irrelevant. Es gibt auch andere deutlich schnellere Matchingalgorithmen, wie z.B. den FLANN-Matcher (Fast Approximate Nearest Neighbor Search Library). Zu finden in [17].

3. Stand der Technik und ähnliche Arbeiten

In diesem Kapitel wird kurz auf Arbeiten und Veröffentlichungen im Internet eingegangen, welche eine ähnliches Thema wie das dieser Arbeit beleuchten.

Der Fokus liegt hier besonders auf den Ergebissen um später zu sehen ob andere Autoren zu den gleichen Resultaten, wie sie im Laufe dieser Arbeit entstanden sind, kommen.

Evaluation of Local Detectors and Descriptors for Fast Feature Matching

In [18] geht es um Detektoren und Deskriptoren zum schnellen Merkmalsvergleich. Der Fokus liegt auf binären Deskriptoren, weil sie eben aufgrund des Vergleichs mittels der Hammingdistanz schneller als herkömmliche Merkmalsverglichsalgorithmen sein sollten.

Es werden folgende Detektoren verglichen: SURF, DoG, FAST, STAR, MSER, BRISK und ORB. Difference of Gaussian (DoG) ist einfach nur als Vergleichswert mit aufgeführt um zu sehen wie viel schneller die anderen Algorithmen arbeiten.

Zum besseren Vergleich werden bestimmte Größen, wie z.B. Repeatability score und Precision-recall, eingeführt.

An der Ergebissetabelle (der Detektoren) ist deutlich zu sehen, welche am schnellsten sind:

Detector	Run time [ms.]	Speed-up [-]	# keypoints
SURF	176	1.9	2 911
DoG	338	1.0	1 552
FAST	**2**	169.0	**5 158**
STAR	17	19.9	849
MSER	60	5.6	483
BRISK	**10**	33.8	1 874
ORB	**7**	48.3	594

Abbildung 3.1.: Laufzeit Detektoren

FAST, BRISK und ORB sind deutlich schneller als der Rest, interessant ist auch die recht unterschiedliche Zahl der gefundenen Merkmale (# keypointes).

Die betrachteten Deskriptoren sind: SIFT, SURF, BRIEF, ORB, LIOP, MROGH, MRRID und BRISK. Auf drei davon wird hier nicht eingegangen, weil sie noch nicht von OpenCV bereitgestellt werden.

In der nächsten Abbildung folgen die Laufzeiten der Deskriptoren:

Descriptor	Run time [ms.]	Speed-up [-]
SURF	117.1	3.83
SIFT	448.6	1.00
BRIEF	**3.8**	118.05
BRISK	**10.6**	42.32
ORB	**4.2**	106.80
LIOP	1 801.1	0.25
MROGH	2 976.8	0.15
MRRID	5 625.1	0.08

Abbildung 3.2.: Laufzeit Deskriptoren

Wieder sind die selben Algorithmen deutlich schneller.

In der letzten Tabelle werden alle Deskriptoren verglichen, dazu wird (für die ersten acht Zeilen) immer der gleiche Detektor benutzt.

Detector	Descriptor	Precision	Recall	MAP
SURF	SURF	0.485	0.513	0.334
SURF	SIFT	0.525	0.533	0.491
SURF	BRIEF	0.517	0.546	0.514
SURF	ORB	0.448	0.470	0.437
SURF	LIOP	0.581	0.597	0.568
SURF	MROGH	0.540	0.567	0.527
SURF	MRRID	0.550	0.569	0.510
SURF	BRISK	0.536	0.553	0.530
BRISK	BRISK	0.504	0.527	0.492
ORB	ORB	0.493	0.495	0.463
FAST	SIFT	0.366	0.376	0.336

Abbildung 3.3.: Deskriptorenvergleich

Hier ist zu sehen, das obwohl manche Algorithmen deutlich schneller waren, sie alle ähnlich große Werte für Precision und Recall erzielen. Wie sich die Zahlen genau zusammensetzen ist der Quelle ([18]) zu entnehmen. (Die Tabllen sind ebenfalls dem Paper entnommen.)

Es wird auch noch kurz auf die Optimierung des Matchings eingangen, das ist aber (wie unter 2.3 beschrieben) in der vorliegenden Arbeit nicht Hauptthema.

A Performance Evaluation of Local Descriptors

In der folgenden Veröffentlichung [19], welche u.a. vom selben Autor wie die erste Quelle des Kapitels stammt, werden nur Deskriptoren und deren Leistung betrachtet.

Es werden wieder Metriken zum Vergleich der verschiedenen Algorithmen aufgestellt. Als Bilddaten werden (wie auch in dieser Arbeit) die Beispieldatensets [20] der University of Oxford verwendet.

Zum Testen werden die Bilder skaliert, gedreht, geglättet, JPEG-komprimiert und die Helligkeit verändert.

Zu jeder der genannten Bildveränderungen gibt es mehrer Graphen. Zur deren Ansicht sollte das Paper selbst [19] herangezogen werden.

Nur die finale Matching Tabelle wird hier zitiert:

Descriptor	recall	1-precision	#nearest neighbor correct matches
GLOH	0.25	0.52	192
SIFT	0.24	0.56	177
Shape context	0.22	0.59	166
PCA-SIFT	0.19	0.65	139
Moments	0.18	0.67	133
Cross correlation	0.15	0.72	113
Steerable filters	0.12	0.78	90
Spin images	0.09	0.84	64
Differential invariants	0.07	0.87	54
Complex filters	0.06	0.89	44

Abbildung 3.4.: Matching Tabelle

GLOH leistet hier die beste Arbeit, auf den letzten Plätzen sind verschiedene andere einfache oder komplexe Filter.

Local Invariant Feature Detectors: A Survey

Eine weiter erwähnenswerte Quelle ist [21]. In dem über 80 Seiten langen Überblick wird auch noch zu Anfang erklärt was Merkmale und Detektoren genau sind und wie sie grundlegendend funktionieren.

Es werden auch ganz einfache Detektoren, wie HARRIS und SUSAN, vorgestellt. Um in das Thema, ohne Grundlagen, einzusteigen ist die Lektüre von [21] sehr empfehlenswert.

Ergebisse bzw. einen zusammenfassenden Überblick über die verschiedenen Detectoren gibt es aber auch noch (in Form einer großen Tabelle, die hier zur besseren Übersicht auf zwei kleinere Tabellen aufgeteilt wurde):

Feature Detector	Corner	Blob	Region	Rotation invariant	Scale invariant	Affine invariant
Harris	√			√		
Hessian		√		√		
SUSAN	√			√		
Harris-Laplace	√	(√)		√	√	
Hessian-Laplace	(√)	√		√	√	
DoG	(√)	√		√	√	
SURF	(√)	√		√	√	
Harris-Affine	√	(√)		√	√	√
Hessian-Affine	(√)	√		√	√	√
Salient Regions	(√)	√		√	√	(√)
Edge-based	√			√	√	√
MSER			√	√	√	√
Intensity-based			√	√	√	√
Superpixels			√	√	(√)	(√)

Abbildung 3.5.: Detektionsbereiche und Invarianzen

In der ersten Tabelle ist zu sehen, welche Detectoren welche Art von Merkmalen und Merkmalsbereichen finden können. Dazu ist aufgezeigt, gegenüber welcher Bildveränderungenen die Detektoren invariant sind. (D.h. welche Veränderungen sind handeln können.)

Feature Detector	Repeatability	Localization accuracy	Robustness	Efficiency
Harris	+++	+++	+++	++
Hessian	++	++	++	+
SUSAN	++	++	++	+++
Harris-Laplace	+++	+++	++	+
Hessian-Laplace	+++	+++	+++	+
DoG	++	++	++	++
SURF	++	++	++	+++
Harris-Affine	+++	+++	++	++
Hessian-Affine	+++	+++	+++	++
Salient Regions	+	+	++	+
Edge-based	+++	+++	+	+
MSER	+++	+++	++	+++
Intensity-based	++	++	++	++
Superpixels	+	+	+	+

Abbildung 3.6.: Bewertung

Im zweiten Teil werden die Detektoren, anhand verschiedener Kriterien, bewertet.

[21] ist gut um sich in das Thema einzuarbeiten, Bildverarbeitungsprofis können aber einen großen Teil der ersten Kaptitel überspringen.

Vergleiche mit anderem Schwerpunkt

Weitre verwandte Arbeiten sind [22] und [23]. In beiden werden Detktoren und Deskriptoren bewertet, allerdings in Betrachtung eines bestimmten Anwendungsfalls. In [22] liegt der Fokus auf dem Verfolgen von sichtbaren Objekten. Die zweite Veröffentlichung bewertet Algorithmen auf der Grundlage 3d-Objekt zu erkennen.

Beide Paper haben nicht ganz direkt viel mit dem Thema dieser Arbeit gemeinsam, sind aber während der Recherche gelesen worden. Sie sollten nicht unerwähnt bleiben, für den Fall das diese Arbeit mit dem Fokus auf einem der beiden Schwerpunkte studiert wird.

Computer Vision Talks

Als letzte artverwandte Arbeit wird hier eine Internetseite aufgeführt. Sie heißt Computer Vision Talks ([24]).

In dem ersten Artikel [25] werden die Merkmalserkennungsalgorithmen, welche OpenCV zu Verfügung stellt verglichen.

Im Gegensatz zu der vorliegenden Arbeit war in 2011 (als der Artikel veröffentlicht wurde) ORB noch nicht implementiert. Deshalb wurden folgende Detektoren verglichen FAST, GoodFeaturesToTrack, MSER, STAR, SIFT und SURF. Dort sind viele Graphen zur Veranschaulichung abbgebildet. Diese werden hier nicht alle aufgeführt, sie können einfach im Internet betrachtet werden.

Im zweiten Artikel werden die verschiedenen Deskriptoren verglichen: [26]. Ebenfalls sind dort, wie im ersten Artikel, viele Ergebnisgrafiken zu sehen. Es werden auch zusätzlich zu BRIEF, ORB, SIFT und SURF auch noch die Deskriptoren LAZY und RIFF beleuchtet. Zur näheren Analyse sollte ebenfalls der Artikel selbst zurate gezogen werden.

Der letzte (hiergenannte) Artikel auf Computer Vision Talks vergleicht kurz die Deskriptoren SURF, FREAK und BRISK ([27]). Neben den Ergebnisgrafiken sind hier auch kurze Auszüge des Quellcodes zu sehen und beschrieben. Dieser kann bei eigener Programmierung eventuell als Inspiration dienen.

Nun sind die Grundlagen für die Auswahl der Daten und Beurteilung der Ergebisse gelegt. Diese Punkte werden im folgenden Kapitel (4) behandelt.

Im übernächsten Kapitel (5) ist die Programmierung des Codes welcher die Ergebnisse liefert erklärt.

4. Daten und Ergebisse

In dem folgenden Kaptitel wird auf die verwendeten Daten (Eingangsbilder und Ausgabedaten der Software) und die Ergebnisse des Vergleichs eingegangen. Hier wurde die größte Eigenleistung erbracht um zu die finalen Resultate zu erreichen.

4.1. Daten

In diesem Unterkapitel wird auf die verwendeten Bilddaten und die Aussgabedaten der Software eingegangen.

4.1.1. Verwendete Bilddaten

Grundsätzlich wurden nur Graubilder benutzt, weil die Detektoren nur Graubilder verarbeiten können. Um besser erkennen zu können was auf den Bilder zu sehen ist werden sie hier jedoch in Farbe dargestellt.

Quelle der Einzelbilder ist der bereitgestellte Download von [28]. Zu finden unter [29].

a1 a2 a3 a4 a5

Abbildung 4.1.: Verwendete Bilder

Des Weiteren wurden die OpenCV Beispieldatensets verwendet. Sie können unter [20] heruntergeladen werden. Zwischen der Aufnahme der Bilder der Sets Baumrinde und Boot wurde gezoomt und die Kamera rotiert. Bei den Bilderserien Graffiti und Mauer wurde der Gesichtspunkt verändert, d.h. der Fotograph hat das selbe Objekt aus verschiedenen Perspektiven aufgenommen. Bei der Szene Leuven wurde die Helligkeit verändert, bei den Bäumen ein Weichzeichnugfilter (bzw. hier mir dem Kameraobjektv absichtlich unscharf gestellt) angewandt und bei den Bildern UBC wurde eine JPEG Kompression verwendet.

Die restlichen Bilder sind im Appendix (A.2) zu sehen.

Baumrinde 1	Baumrinde 2	Baumrinde 3	Baumrinde 4	Baumrinde 5	Baumrinde 6

Abbildung 4.2.: Set Baumrinde

4.1.2. Ausgabe der Software

Die Ausgabe der Software sieht zu jedem Paar aus Detektor und Deskriptor folgendermaßen aus:

pi/st	an/co	t_dec	#kp	t_des	#des	t_mat	#gmatch	per	matrat
a/a1	-	1	566	7	528				
a/a1	0	1	566	6	528	11	528	1	1
a/a1	5	1	445	4	429	6	295	0.963	0.662
a/a1	10	0	461	5	455	5	71	0.915	0.143
a/a1	15	1	449	4	443	7	12	0.833	0.0226
...									

Tabelle 4.1.: Softwareausgabe

rep	correspCount
1	528
0.965	414
0.96	437
0.923	409
...	

Tabelle 4.2.: Softwareausgabe (Fortsetzung)

Unter **pi/st** wird aufgeführt, welches Bild oder Datenset verwendet wurde. Z.B. **a/a1** steht für das erste Bild im Ordner **a**. Ein ganzes Wort wie **bark** steht für ein Datenset.

In der Spalte mit Namen **an/co** ist der (Dreh-)Winkel oder Koeffizient bei einem Bild aufgeführt. Bei einem Datenset steht an der Stelle immer eine zweistellige natürliche Zahl zwischen **11** und **15** diese gibt wieder welches mit welchem Bild der Serie verglichen wurde. **12** steht dafür, dass Bild **1** mit Bild **2** verglichen wurde. (Die Zeile mit **11** ist für einen Testdurchlauf, um zu testen ob die Algorithmenkombination überhaupt funktioniert. Denn manchmal findet die Kombination auch bei kleinen Änderungen keine Matches mehr.)

Steht an der Stelle ein **-** handelt es sich um eine Kontrollzeile. In dem Programmdurchlauf wurde nur detektiert und beschrieben (mit Hilfe des Deskriptors), aber eben noch nicht verglichen, weil dazu nur ein einzelnes Bild nicht reicht.

t_dec, **t_des** und **t_mat** sind die benötigten Zeiten zum Detektieren, Deskriptieren und Matchen der Merkmale in Millisekunden.

Unter **#kp** und **#des** ist die Anzahl der gefundenen Merkmale und zugehörigen Deskriptoren angegeben. Hier ist die erste Zahl oft größer da Punkte zusammen fallen können und deshalb nur einmal beschrieben werden müssen. (Das ist aber Aufgabe des Deskriptors und wird damit auch in dessen Zeit mit einberechnet.)

Mit Zahl unter **#gmatch** wird die Anzahl der Matches ausgegeben. Das sind Paare von Punkten in den beiden Bildern die einer bestimmten Güte genügen. Hier wird der Abstand eines Deskriptors im Ursprungsbild zu seinen beiden nächsten Nachbarn im neuen Bild berechnet, nur wenn der Abstand zum nächsten Nachbarn 0.6 oder weniger als der Abstand zum zweitnächsten Nachbarn ist wird das Punktepaar nicht verworfen. So wird im Zweifel (die beiden nächsten Nachbarn sind ca. gleich weit weg), das Punktepaar nicht verwendet um Fehler zu vermeiden. Warum genau 0.6 gewählt wird ist in [2, Seite 20] nachzulesen.

per sind die Prozent der richtigen Matches, die aus dem Urbild errechnet wurden, geteilt durch die Matches welche der Algorithmus im veränderten Bild gefunden hat (**#gmatch**).

Um das Verhältnis der Paare (**matrat** für Matching Ratio) zu berechnen wir zu erst ein Hilfswert benötigt.

Er gibt den Prozentsatz der gefundenen Paare geteilt durch das Maximum der zu findenden Paare an. Ergo wird der Wert von **#gmatch** durch das Minimum der Anzahl der gefundenen Merkmale in beiden Bildern geteilt. Hier werden die Verhältnisse, welche am Ende betrachtet werden von der Anzahl der gefundenen Punkte gelöst so können, auch die verschiedenen Algorithmen gut verglichen werden, obwohl sie mit unterschiedlich vielen Merkmalen arbeiten.

Um die Matching Ratio zu berechnen wird der Hilfswert noch mit dem Wert **per** multipliziert.

Die Werte in den letzten beiden Spalten der Aussgabe (**rep** und **correspCount**) werden mit Hilfe der OpenCV Funktion **evaluateFeatureDetector(...)** errechnet. ([30])

Diese Funktion errechnet aus den beiden Bildern, den Merkmalen und der Homographiematrix (**H**) die Reproduzierbarkeit (**rep**) und die Anzahl der Korrespondenzen (**correspCount**).

Die Matrix (**H**) ist eine Abbildungsmatrix, die aus den Punktekorrespondenzen, errechnet wird. Sie beschreibt wie das eine Bild zu dem anderen Bild liegt. Es werden aber nur Rotation und Translation berücksichtigt. Für die Einzelbilder wird die Matrix (**H**) in der Software ermittelt, sofern mindestens für Korrespondenzen gefunden wurden. Wenn das nicht der Fall ist kann die Matrix (**H**) nicht berechnet werden und damit auch die Funktion **evaluateFeatureDetector(...)** nicht angewandt werden. Für die Datensets wird die Matrix (**H**)einfach eingelesen, weil sie mit den Datensets zur Verfügung gestellt wird.

Die Reproduzierbarkeit (**rep**) gibt wieder wie hoch die Wahrscheinlichkeit ist das ein Merkmal in anderen Bildern der Szene wiedergefunden wird.

Die Anzahl der Korrespondenzen (**correspCount**), die es wirklich gibt ist praktisch immer größer als die gefundene Zahl der Korrespondenzen (**#gmatch**), außer wenn ein Bild mit sich selbst verglichen wird. (Dann ist sie gleich groß.)

4.2. Ergebisse

In diesem Unterkapitel werden die Ergebnisse, der Softwareausgabe, ausgewertet. Praktisch alle Resultate werden als Diagramme präsentiert und dann beschrieben.

4.2.1. Detektoren im Vergleich

Um alle Detektoren best möglich mit einander vergleichen zu können wurden für dieses Unterkapitel nur Daten verwendet, bei denen die verschiedenen Detektoren mit SURF als Deskriptor benutzt wurden.

Rotation

Als erstes wird hier die Rotation behandelt. Dazu folgt das erste Diagramm:

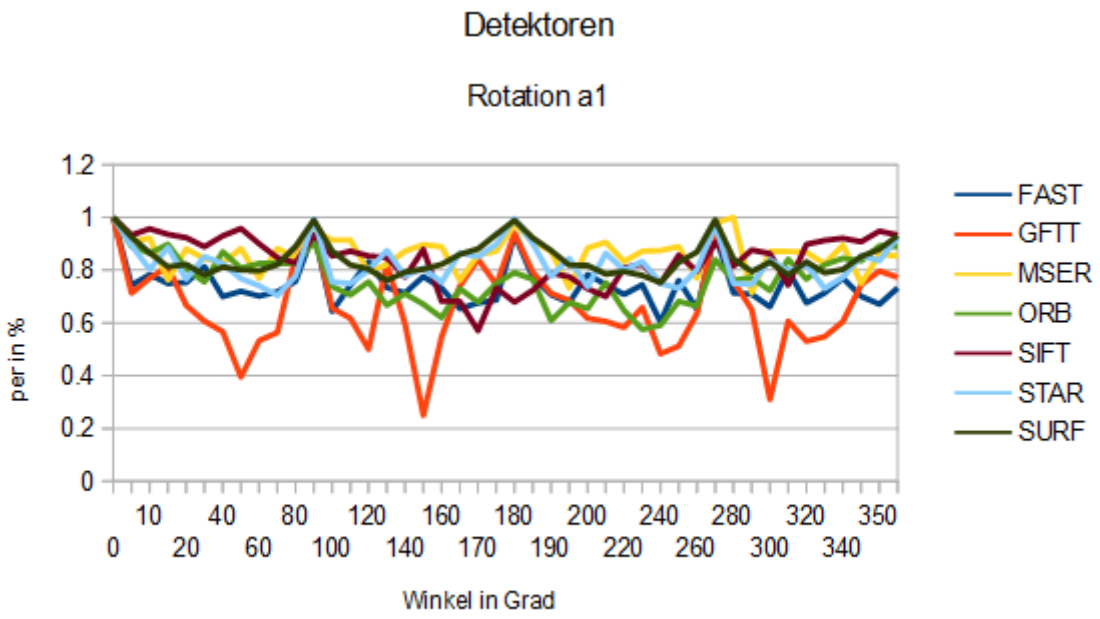


Abbildung 4.3.: Rotation a1

Hier ist der Wert **per** (Prozensatz der richtigen Matches, 4.1.2) in Abhängigkeit des Rotationswinkels angegeben. Es ist (wie erwarten) zu sehen das alle Detektoren in den Bereichen um Vielfache von 90 Grad gut arbeiten und dazwischen schlechter. Hier fällt nur GFTT negativ auf, der Rest verhält sich recht ähnlich. MSER noch am besten und ORB am zweitschlechtesten.

Es wurde hier das erste Bild **a1** verwendet, da in dessen Diagramm am meisten zu sehen ist. Die Diagramme zu den anderen Bilder sind im Anhang zu finden (A.3).

Ebenfalls sind in dem Diagramm zu **a1** recht gut Unterschiede zu sehen:

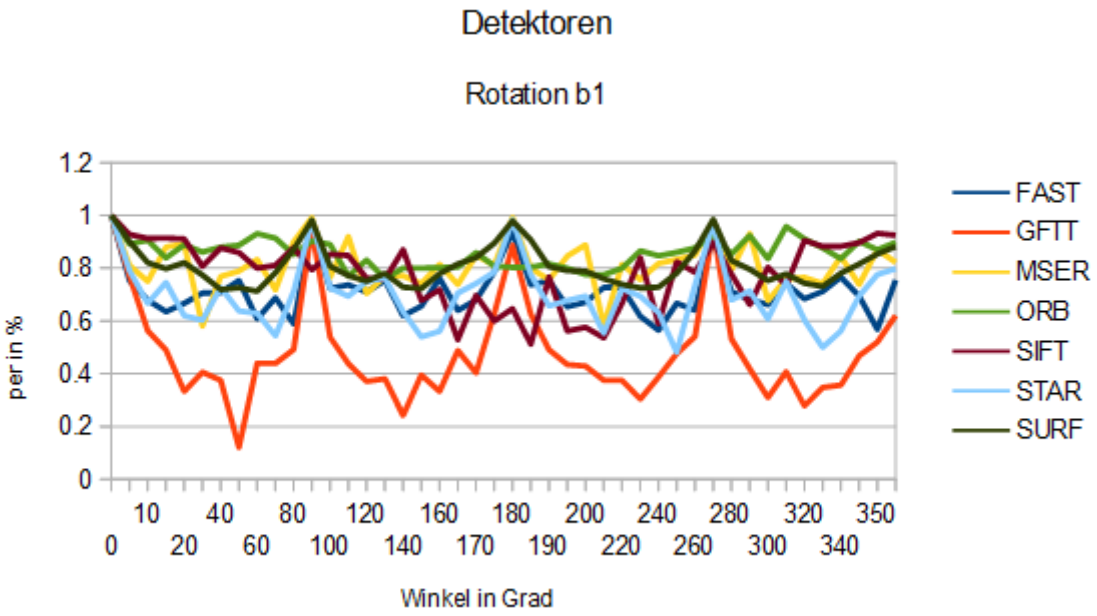


Abbildung 4.4.: Rotation b1

Auch hier fällt GFTT negativ auf. ORB ist mit am besten und STAR am zweitschwächsten.

Für die weiteren Punkte werden jetzt der Übersichtlichkeit halber nur noch die Bilder **a1** und **b1** analysiert.

Skalierung

Es folgen die beiden Diagramme zum Vergleich der Detektoren wenn die Bilder vergrößert oder verkleinert werden:

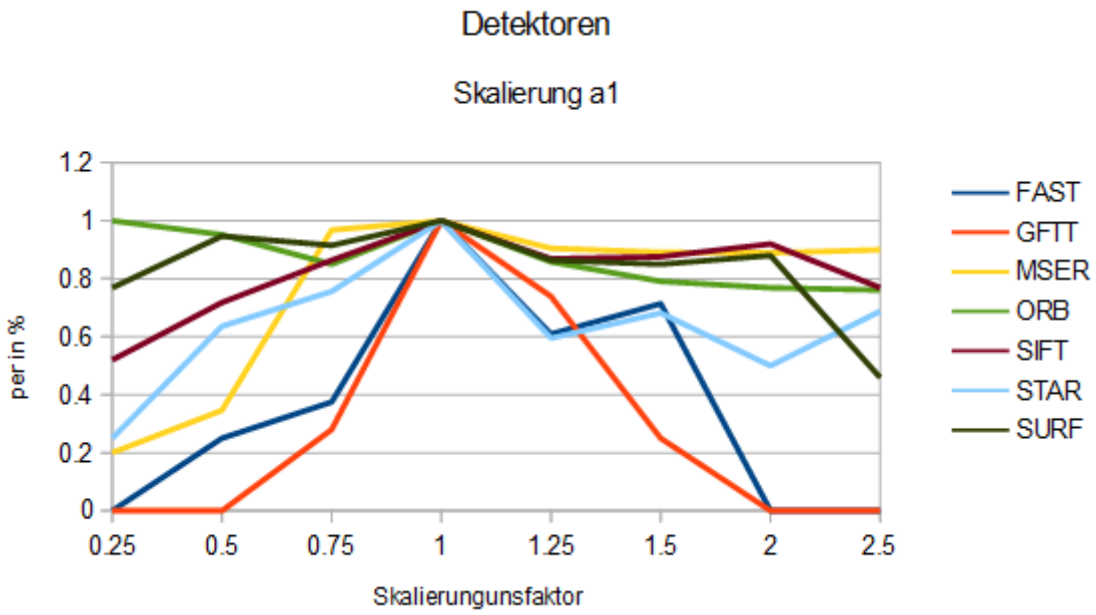


Abbildung 4.5.: Skalierung a1

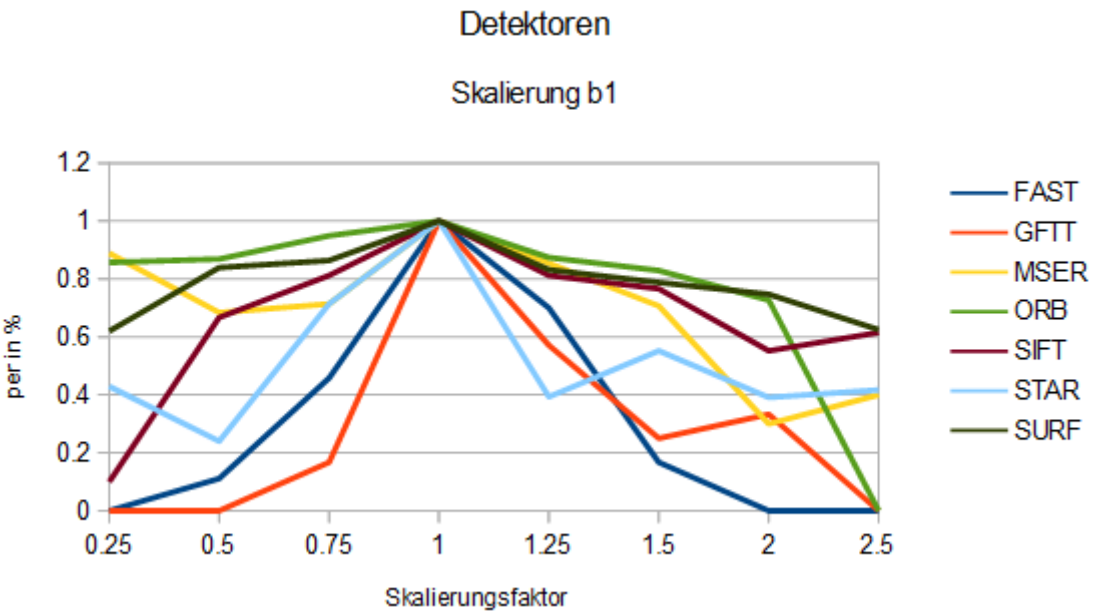


Abbildung 4.6.: Skalierung b1

Zu sehen ist das GFTT und FAST am schlechtesten abschneiden, ORB und SURF am besten.

Helligkeitsveränderung

Nun werden die Auswirkungen gezeigt, wie sich die Veränderung der Helligkeit auf das Wiederfinden der Merkmale auswirkt.

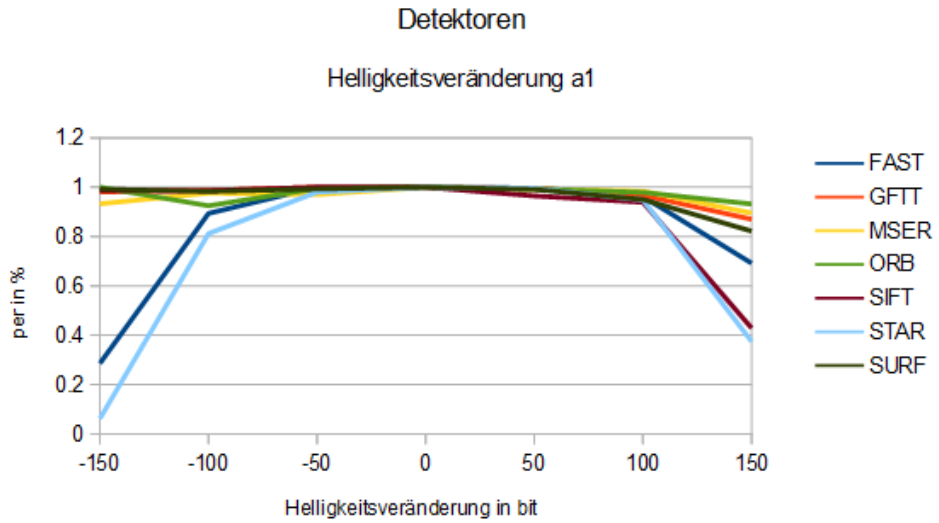


Abbildung 4.7.: Helligkeitsveränderung a1

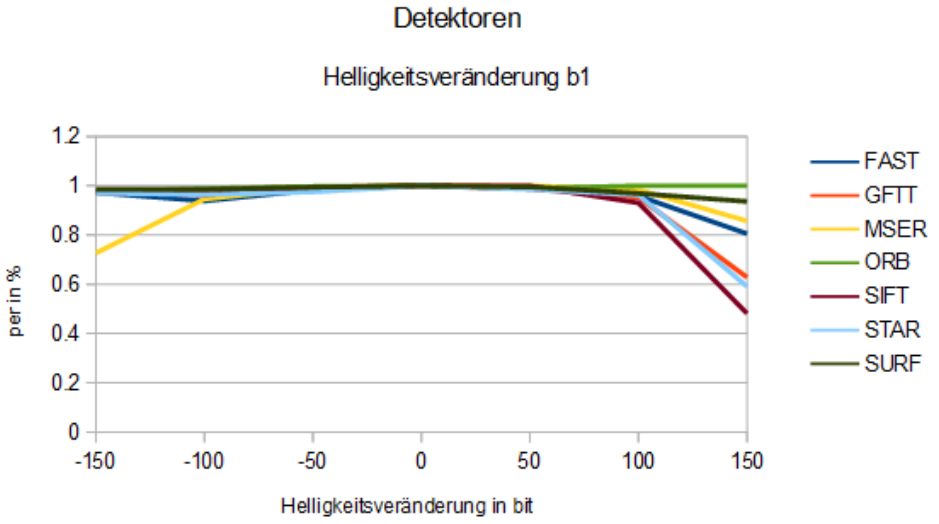


Abbildung 4.8.: Helligkeitsveränderung b1

Wenn Werte bis zu 100 bit auf die Werte des Bildes addiert oder von ihnen subtrahiert werden (bei 256 bit Bildtiefe) erkennen alle Algorithmen noch einen Großteil der Merkmale wieder (über 90% werden erkannt). Nur STAR schwächelt an einer Stelle (im ersten Bild dieses Unterkapitels zu sehen 4.7) und erkennt aber immer noch 80% der Merkmale wieder.

Bei Veränderungen der Helligkeit um über 100 bit, fangen FAST, STAR, aber auch SIFT an nicht mehr den Großteil der Merkmale zu erkennen.

Unschärfe

In den nächsten beiden Abbildungen ist zu sehen wie sich das Glätten mit einem Gaußfilter, auf das Wiederfinden der Merkmale auswirkt. Mit zunehmender Filtergröße wird das Bild schnell unschärfer und so fallen die Kurven alle recht erwartungsgemäß ab.

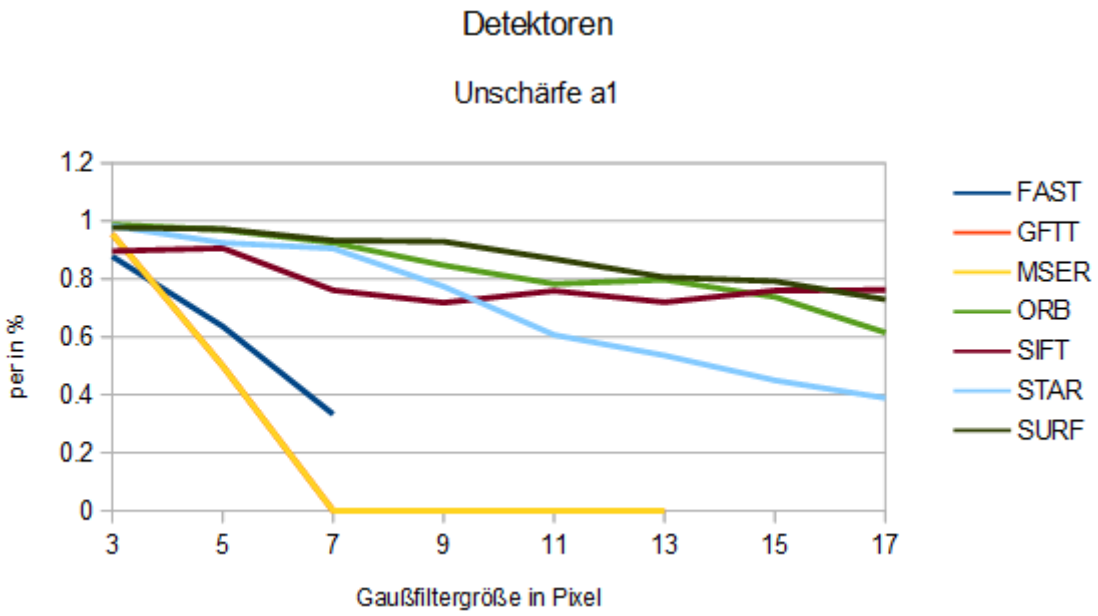


Abbildung 4.9.: Unschärfe a1

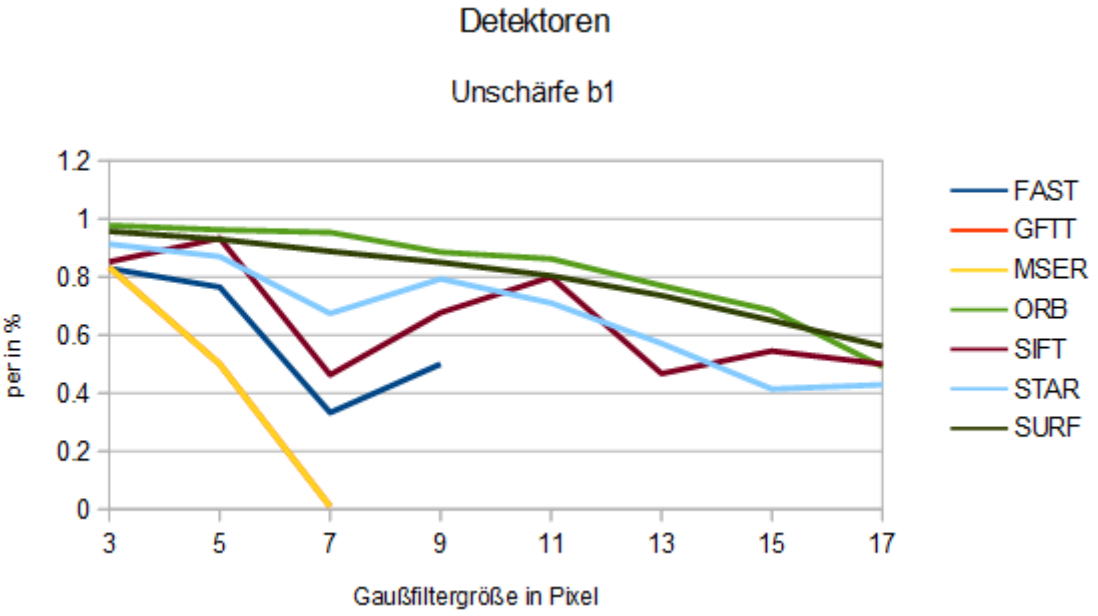


Abbildung 4.10.: Unschärfe b1

Die besten Detektoren SIFT, SURF und auch ORB finden aber auch bei einer Gaußfiltergröße von 17 Pixeln noch die Mehrheit der Merkmale wieder.

FAST und MSER geben letztlich ganz auf. D.h. es kann kein Wert mehr für **per** errechnet werden, da gar keine guten Matches (**#gmatch**) mehr gefunden werden und somit durch

Null geteilt werden müsste (wie in 4.1.2 beschrieben). Der Unterschied von keinem Wert zu einer Null ist, das eben bei einer Null noch Merkmale im aktuellen Bild gefunden werden, aber sie aber eben zu keinem Merkmal (im Urbild) passen.

Der kleine Anstieg in 4.10, zwischen den Werten sieben und neun, ist höchstwahrscheinlich damit zu erklären das die Größe der Gaußfiltermaske zum Teil zufällig gut mit den Masken der Detektoren harmoniert. Aber das ein Phänomen das nicht ausgenutzt werden kann, weil es sicher auch von den Bilddaten abhängt (in dem anderen Diagramm taucht es nicht auf) und auch ein unscharfes Bild nie aussieht als wäre es mit einem perfekten Gaußfilter geglättet worden. (Der Filter simuliert hier nur Unschärfen.)

Datensets

Nun folgt ein sehr entscheidendes Kapitel, weil mit den Datensets auch komplett andere Bildveränderungen (als in den letzten Unterkapiteln beschrieben) analysiert werden können.

Als Wert auf der Y-Achse wird hier **rep** statt **per** verwendet, da der Wert (**per**) eben dadurch, dass die Bilder der Sets nicht im Programm modifiziert werden nicht errechnet werden kann. Da für die Sets wird die Homographiematrix (**H**) aber mitgeliefert kann der Wert **rep** gut verglichen werden. Unter 4.1.2 sind die Werte erklärt worden.

Die Sets werden hier in einer Reihenfolge betrachtet, die es erlaubt die Datensets mit ähnlichen Veränderungen direkt nacheinander zu abzuhandeln. Dazu und auch wie sich die Bilder innerhalb der Sets unterscheiden kann unter 4.1.1 nachgelesen werden.

Zu Anfang wird die Bilderserie Leuven betrachtet. Hier wurdehauptsächlich die Heilligkeit verändert.

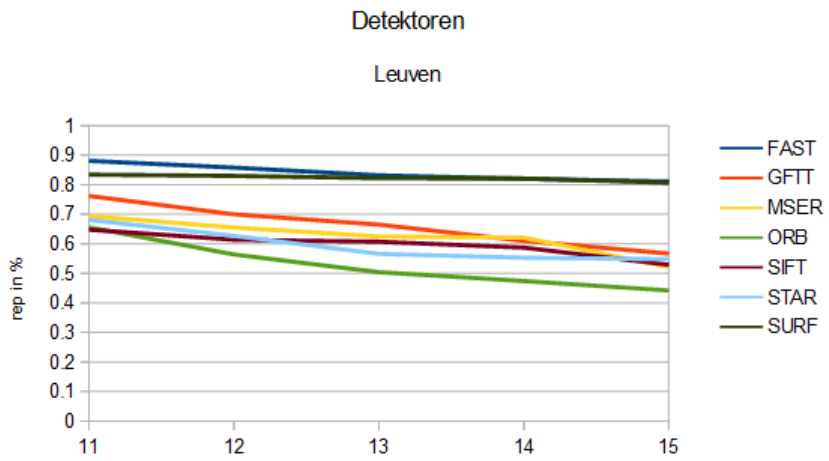


Abbildung 4.11.: Leuven

In dem Diagramm ist nicht viel Neues zu sehen, weil die Helligkeitsänderung weniger als die Eigenen in 4.2.1 beträgt.

Bei den folgenden Serien Baumrinde und Boot wurde zwischen den Bildern gezoomt und die Kamera rotiert. Da die Bildmodifikationen ähnlich sind werden die beiden Sets hier zusammen behandelt.

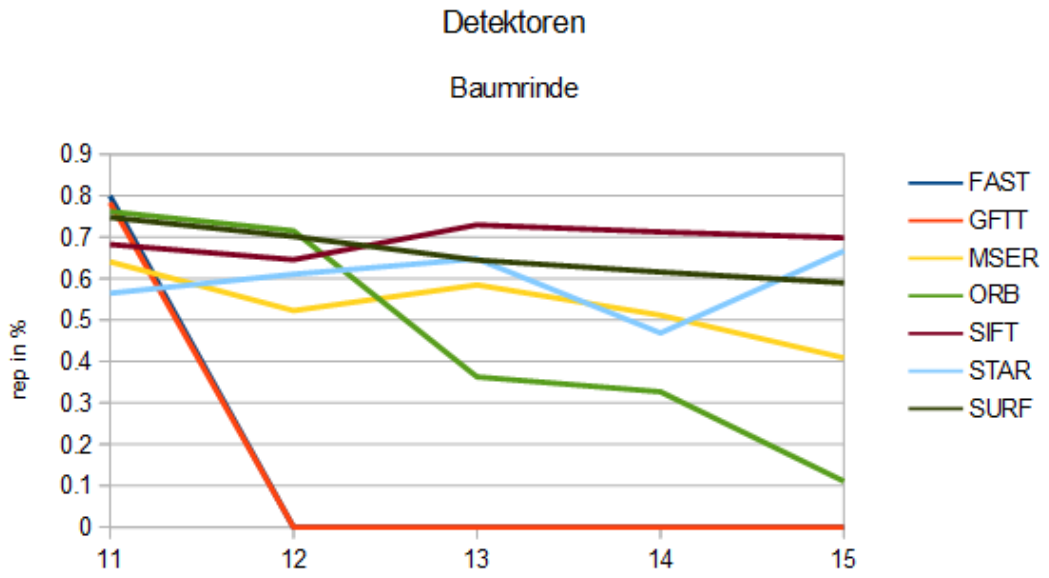


Abbildung 4.12.: Baumrinde

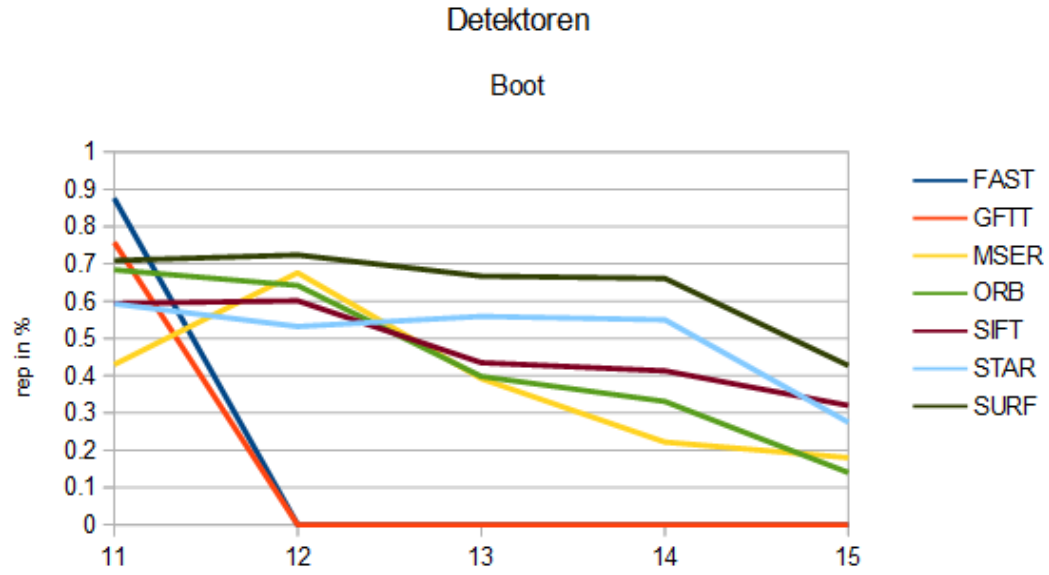


Abbildung 4.13.: Boot

Die beiden Diagramme geben ein im Wesentlichen ähnliches Bild wieder. FAST und GFTT können die großen Rotationen gar nicht handeln. ORB und MSER kommen mittelmäßig bis schlecht mit den Serien klar. Am besten schneiden SURF, SIFT und STAR (in dieser Reihenfolge) ab.

Bei dem Set der Bäumen wurde das Kameraobjektv so verstellt, das der Großteil der späteren Bilder unscharf ist. Das wirkt letztlich wie ein Weichzeichnugfilter.

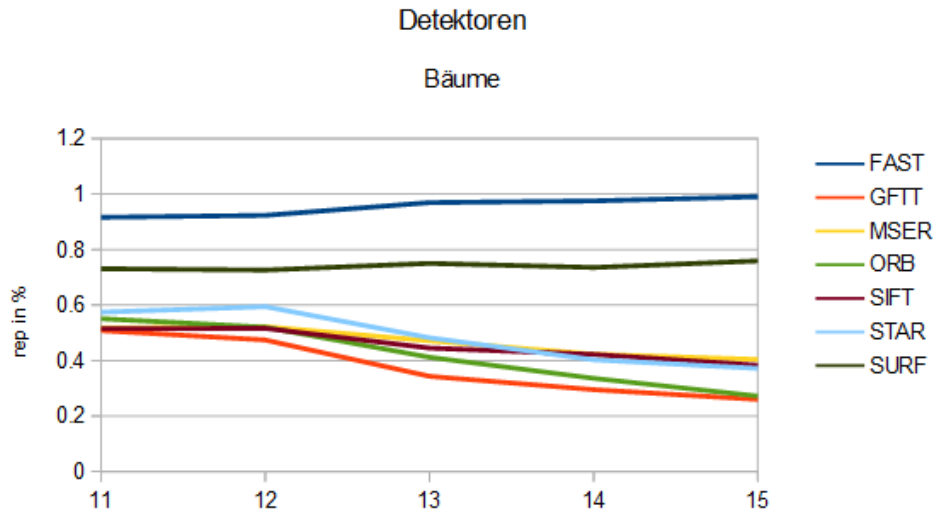


Abbildung 4.14.: Bäume

FAST und SURF kommen in diesem Fall am besten mit der Unschärfe der Bilder zu recht. Die Unschärfe fällt in der Bilderserie wesentlich weniger entscheidend aus, als in 4.2.1. Wenn ein möglichst guter Algorithmus zum kompensieren von Unschärfe gesucht wird, werden am besten SIFT, SURF und auch ORB, die sich schon in 4.2.1 als am geeigentsten dafür herausgestellt haben, verwendet.

Bei den Bilderserien Graffiti und Mauer wurde der Gesichtspunkt verändert, d.h. in diesen Fällen der Fotograph hat sich zum aufgenommenen Objekt bewegt und es in einem immer flacheren Winkel aufgenommen. Bei Betrachtung der beiden Bilderserien unter A.2 wird die Veränderung schnell klar.

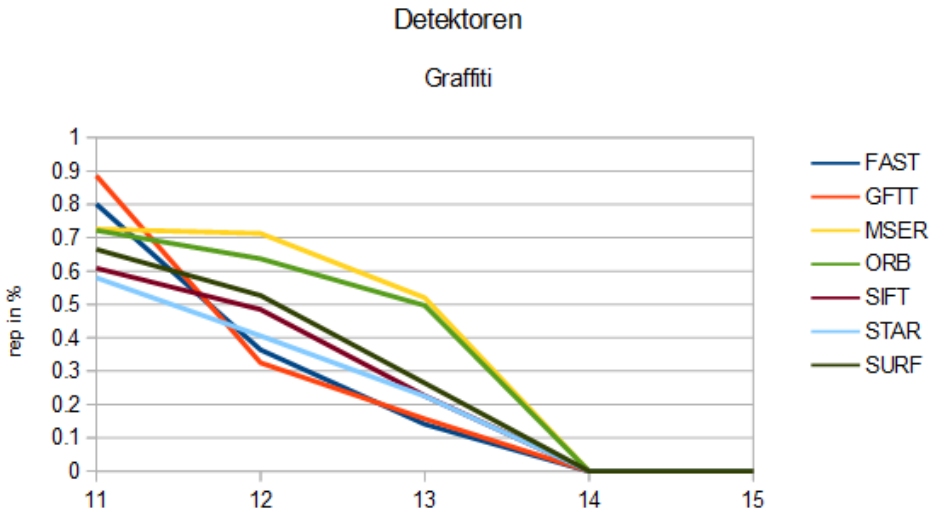


Abbildung 4.15.: Graffiti

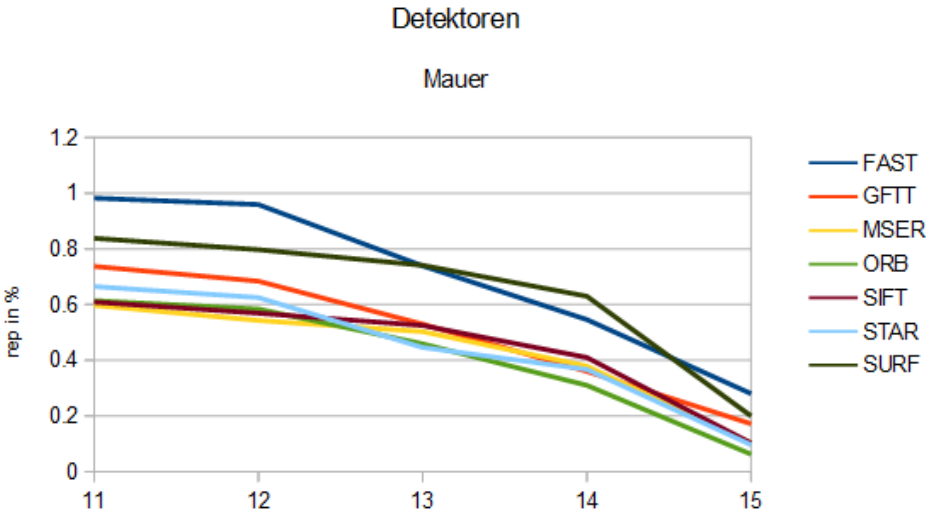


Abbildung 4.16.: Mauer

Die Verläufe der beiden Diagramme sind ähnlich. Mit der Fortgang der Serie sinkt die Wahrscheinlichkeit das die Merkmale wieder gefunden werden.

In dem einen Fall stechen FAST und SURF im anderen MSER und ORB etwas positiv heraus. Bei den deutlichen Veränderungungen in den letzten Bildern stoßen alle Detektoren an ihre Grenzen. Bei der Betrachtung der Bilderserien (insbesondere bei Graffiti) schafft es auch das menschliche Gehirn nicht sofort die Merkmale zu verknüpfen.

In der Bilderserie UBC wurde eine JPEG Kompression verwendet.

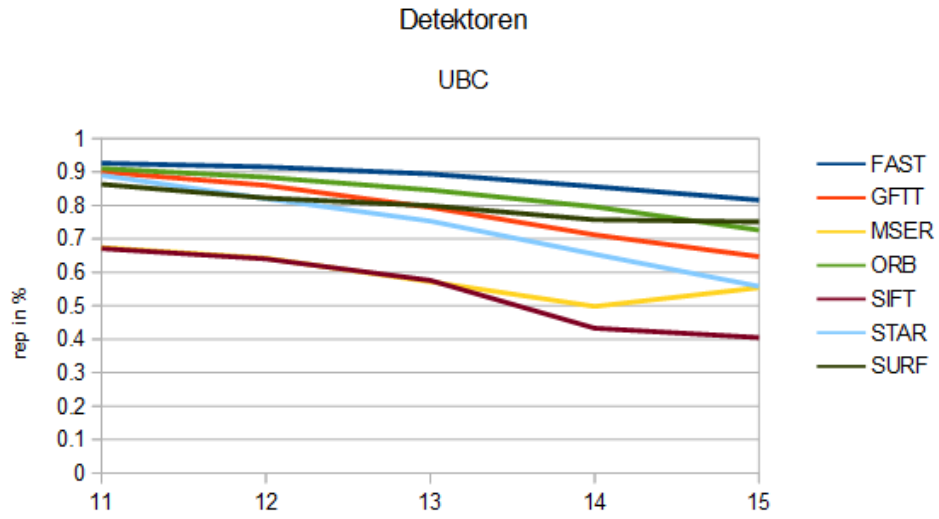


Abbildung 4.17.: UBC

Zu erkennen ist das FAST am besten abschneidet, weil ihm die Kompression sehr in die Hände spielt. Die beiden Verfahren (FAST Detektor und JPEG Kompression) verwenden ähnliche Methoden, wie z.B. das benachbarte Pixel zusammengefasst werden.

MSER und SIFT scheiden hier am schlechtesten ab.

Zeit

Als letztes in diesem Unterkapitel wird noch ein Diagramm zu Detektionszeit angeführt. Hier wurden einfach die Zeiten für ein rotierendes Bild aufgetragen. Die Detektionszeiten eines einzelnen Detektors untererscheiden sich von sich selbst meist nur marginal. Von den anderen aber sehr deutlich.

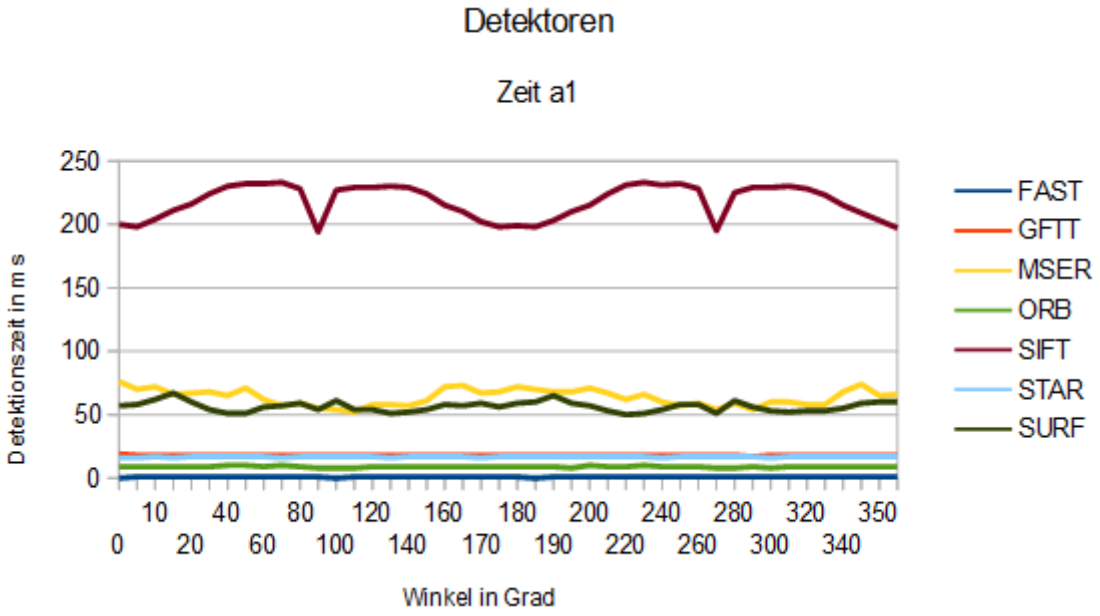


Abbildung 4.18.: Zeit

In dem Diagramm ist zusehen, das SIFT ganz deutlich am langsamsten ist. MSER und SURF sind deutlich schneller, aber immer noch langsam im Vergleich zu FAST, ORB, STAR und GFTT. Hier muss aber auch noch erwähnt werden das ORB und ganz besonders FAST noch mal schneller als STAR imd GFTT sind.

Insgesamt kann festgehalten werden, das SIFT sicher nicht der beste Detektor ist, weil er deutlich langsamer arbeitet als die anderen aber nicht entscheidend bessere Werte an anderer Stelle vorweisen kann. SURF und MSER liefern meist eine recht gute Leistung ab und sollten, wenn etwas genaueres als FAST oder ORB gebraucht werden benutzt werden. STAR und GFTT sind zwar recht schnell liefern aber praktisch keine besseren Resultate als FAST und ORB. Sie sind (aufgrund der hier verwendeten Daten) am schwächsten einzustufen. FAST und ORB sind sehr schnell. Aber gerade FAST liefert oft auch keine brauchbaren Resultate. Wenn Geschwindigkeit im Mittelpunkt steht und die Veränderungen in den Folgebildern eher klein sind sollte er auf jeden Fall getestet werden. ORB ist so etwas wie der Gewinner er ist am zweitschnellsten und sticht bei den Resultaten nicht öfter negativ als positiv heraus.

Wie praktisch immer, in der Bildverarbeitung, hängt der idele Merkmalsdetektor vom Anwendungsfall ab. Hoffentlich kann mit Hilfe des letzten Kapitels der passende Detektor einfacher gefunden werden, oder zumindest schon einige (ohne viel Vorarbeit) ausgeschlossen werden.

4.2.2. Deskriptoren im Vergleich

Wie auch im letzten Unterkaptitel wurden alle Deskriptoren mit einem Detektor (wieder SURF) kombiniert um einen möglicht fairen Vergleich ziehen zu können.

Rotation

Am Anfang werden die Deskriptoren auf ihre Robustheit gegen über Rotation betrachtet.

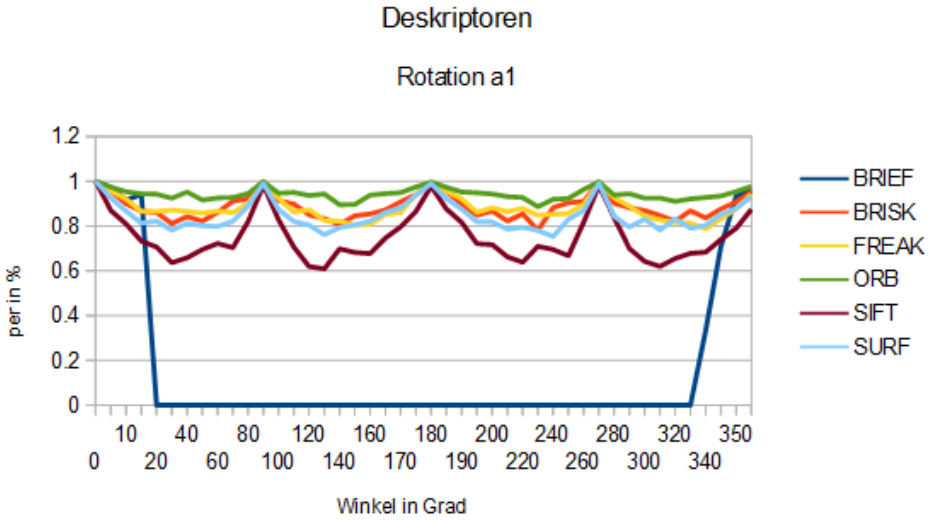


Abbildung 4.19.: Rotation a1

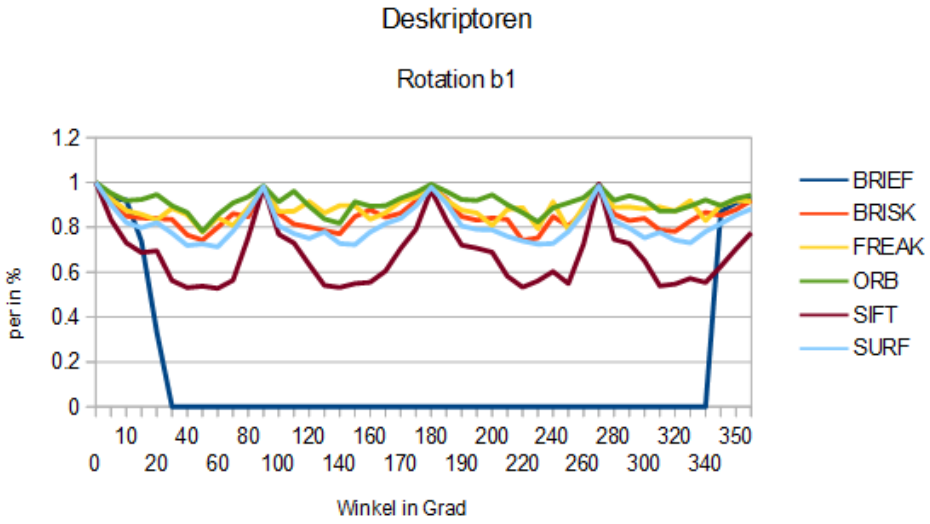


Abbildung 4.20.: Rotation b1

Die beiden Diagramme geben ein ähnliches Bild wieder. BRIEF versagt auf ganzer Linie (ab einem Winkel von über 30 Grad), SIFT ist der zweitschlechteste Deskriptor, aber deutlich besser als BRIEF, ORB ist am besten. Der Rest liegt im guten Mittelfeld.

Skalierung

Die beiden folgenden Abbildungen, stellen den Einfluss von Skalierung auf die Wiedererkennung der Merkmale dar.

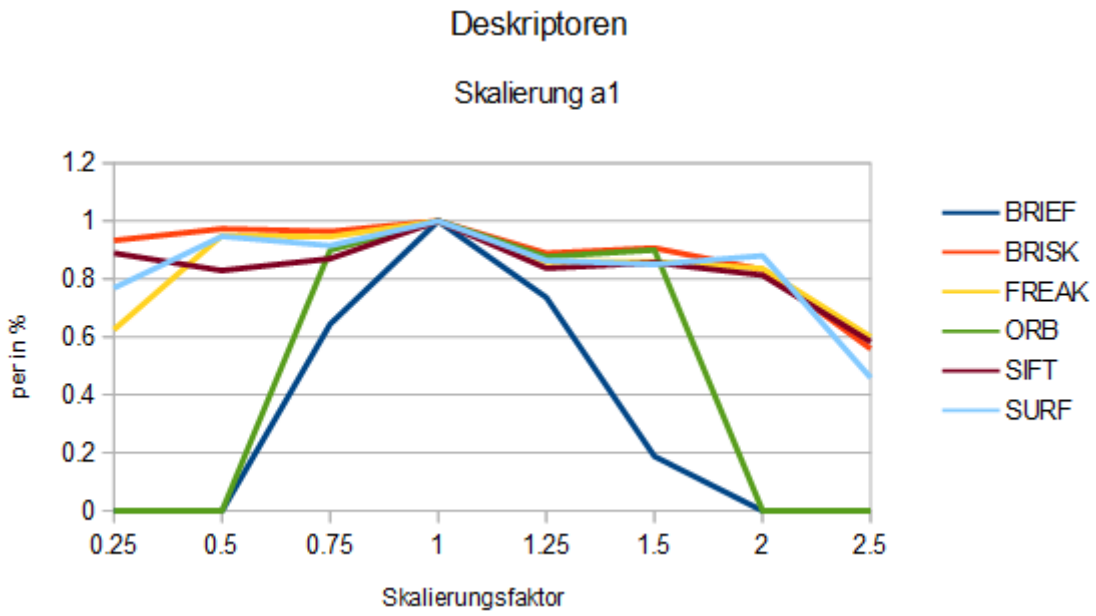


Abbildung 4.21.: Skalierung a1

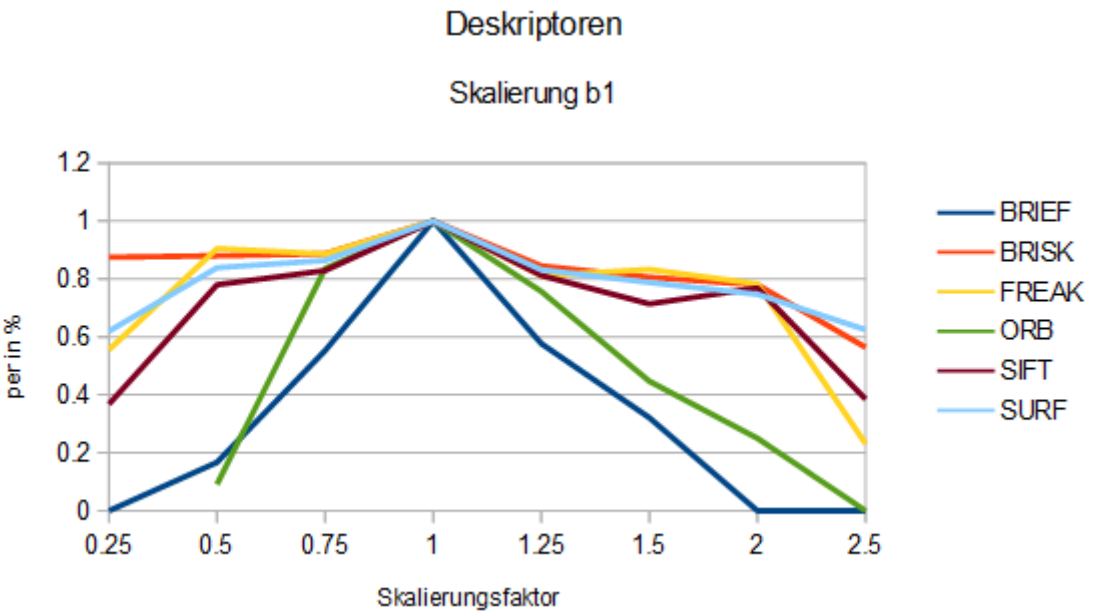


Abbildung 4.22.: Skalierung b1

Die Ergebisse der beiden verschiedenen Bilder unterscheiden sind kaum. BRIEF schneidet wieder am schwächsten ab, ORB ist Zweitletzer, aber auch diese Beiden (besonders ORB) können kleine Skalierungunsunterschiede handeln. BRISK schlägt sich hier am besten. Auch bei großen Skalierungsänderungen findet es den Großteil der Merkmale wieder.

Helligkeitveränderung

Wie in den folgenden Bildern zu sehen, können Helligkeitsveränderungen von allen Deskriptoren gut kompensiert werden:

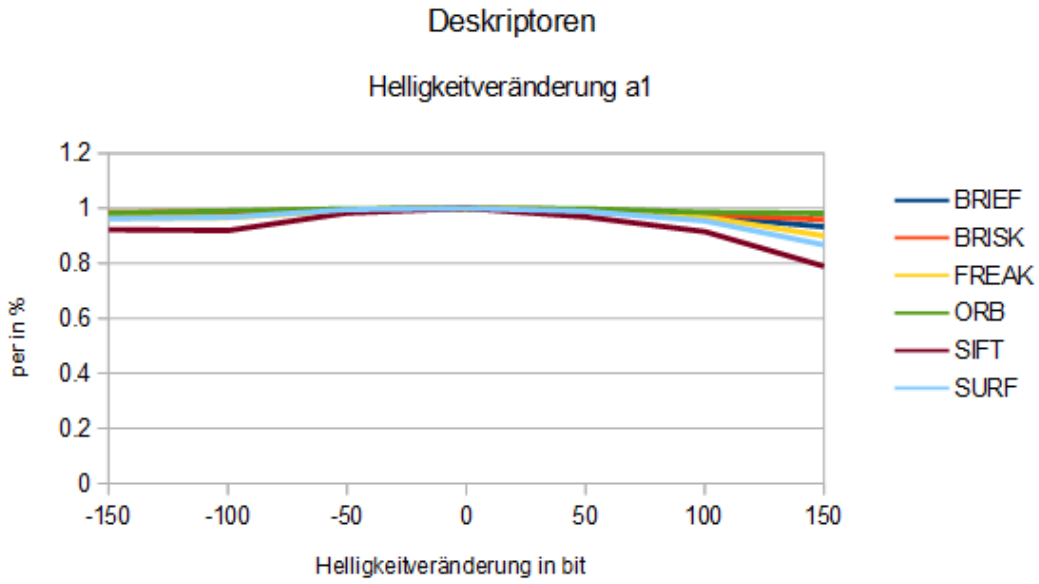


Abbildung 4.23.: Helligkeitsveränderung a1

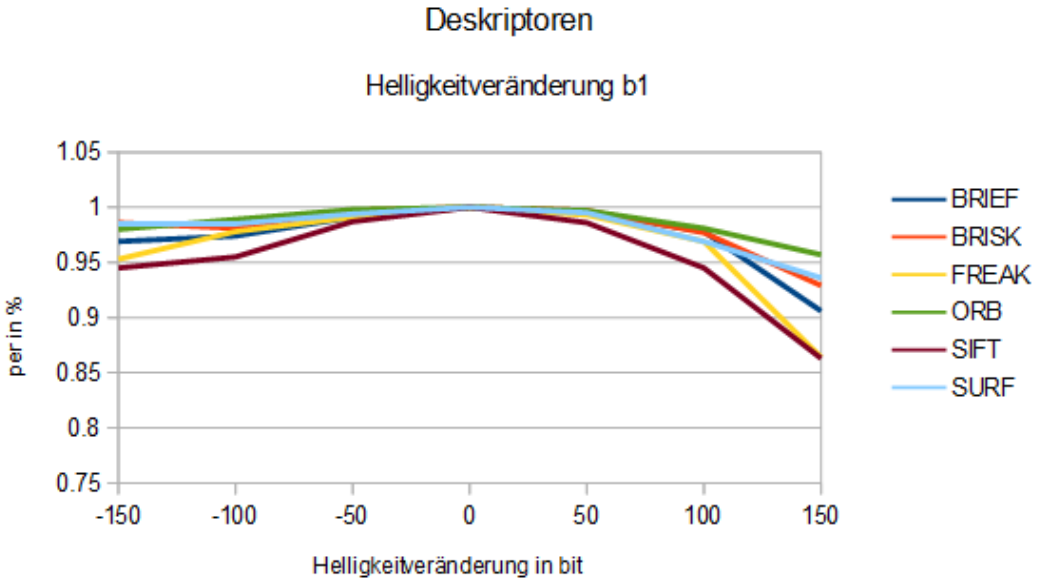


Abbildung 4.24.: Helligkeitsveränderung b1

SIFT und FREAK schneiden etwas schlechter ab als der Rest. ORB ein klein wenig besser.

Unschärfe

Die nächsten beiden Abbildungen stellen die Auswirkungenen von Unschärfe auf die Arbeit der Deskriptoren dar:

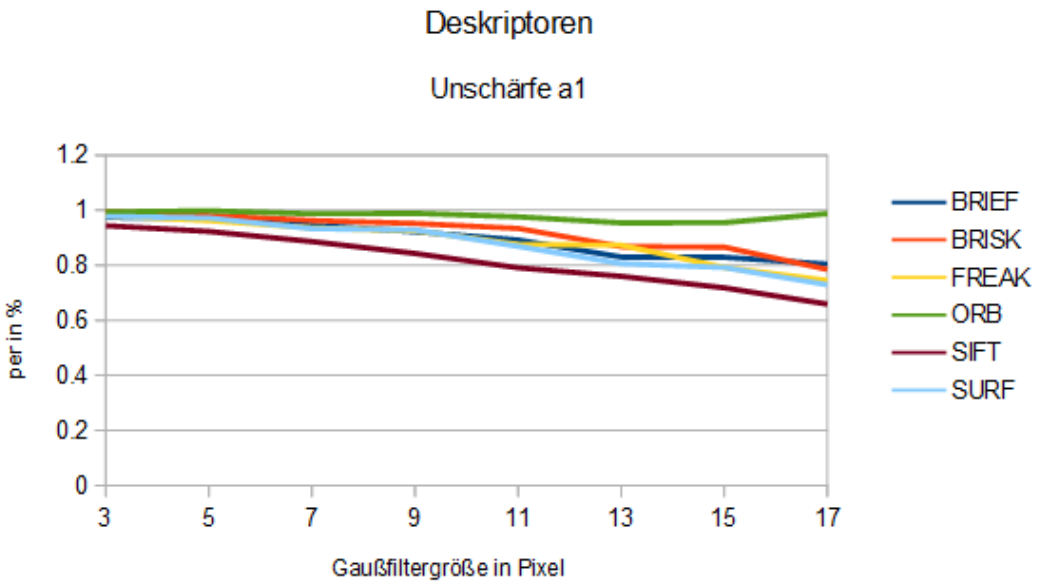


Abbildung 4.25.: Unschärfe a1

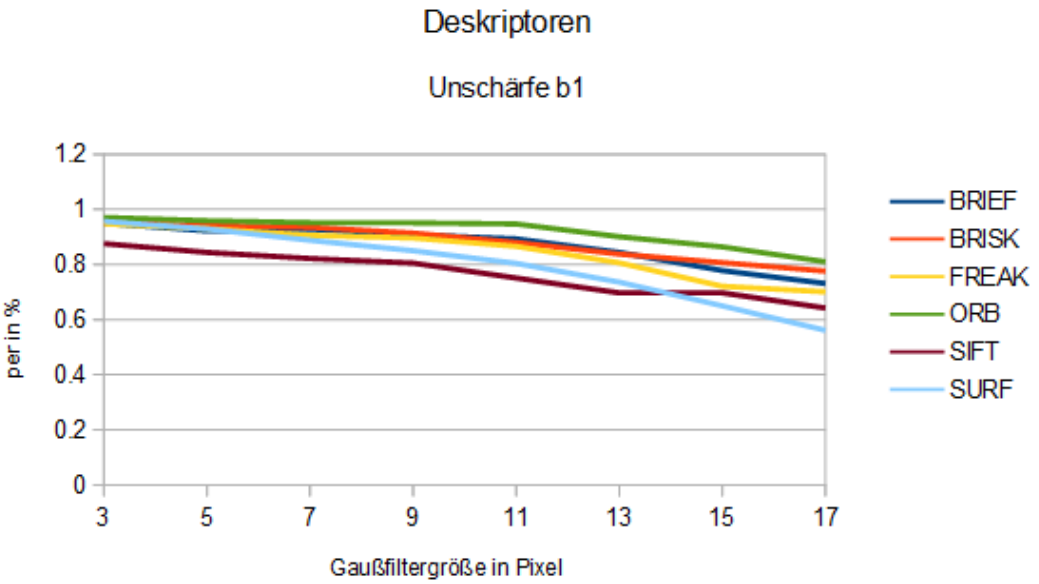


Abbildung 4.26.: Unschärfe b1

Die Linien fast aller Deskriptoren fallen (erwartungsgemäß) stetig ab. Kein Deskriptor liefert richtig schlechte Ergebnisse. SIFT und SURF schneiden noch am schwächsten ab. ORB ist (wieder) am besten.

Datensets

Nun werden die Bilderserien als Testfälle für die verschieden Deskriptoren verwendet. Als Wert auf der Y-Achse wird hier wieder **rep** statt **per** verwendet, wie schon in 4.2.1 beschrieben.

In der Szene Leuven wurde die Helligkeit über die Bilder hinweg verändert.

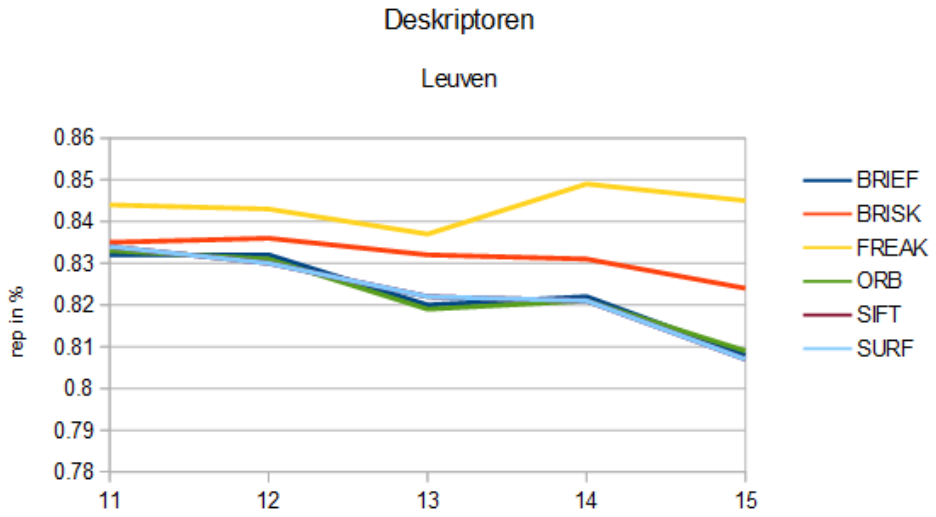


Abbildung 4.27.: Leuven

Hier wurde die Skalierung der Y-Achse gestreckt, damit die Unterschiede der verschiedenen Deskriptoren besser zu erkennen sind. Alle schneiden in etwas gleich gut ab. Nur FREAK etwas besser.

In den Bildsets, welche den folgenden Diagrammen zugrunde liegen wurde, in die Bilder hineingezoomt und die Kamera deutlich rotiert.

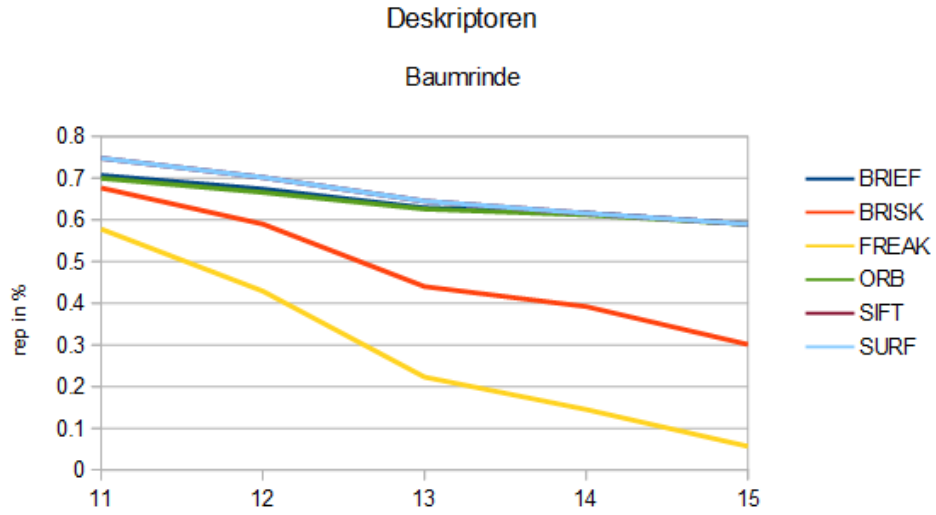


Abbildung 4.28.: Baumrinde

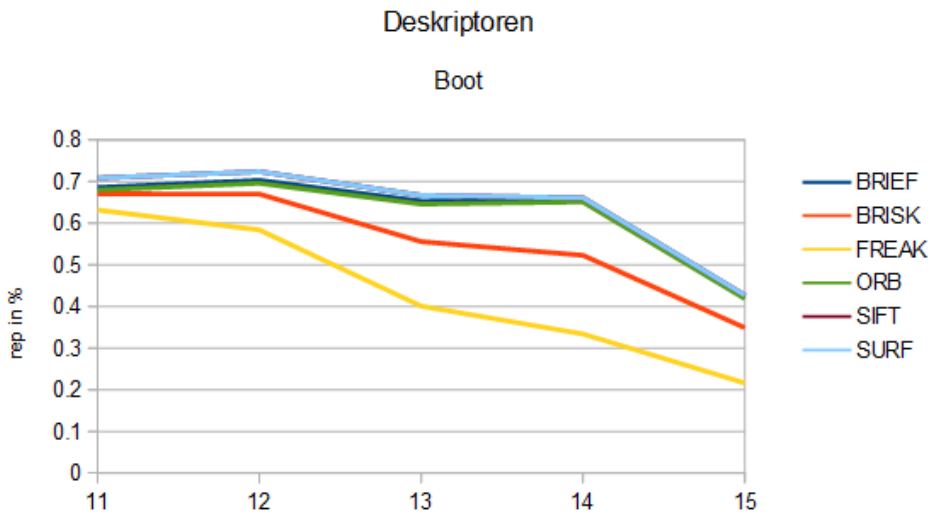


Abbildung 4.29.: Boot

Die Verläufe der Diagramme sind wieder ähnlich. Nur da bei den Bildern der Baumrinde die Kamera mehr rotiert wurde, trennen sich die Kennlinien der Deskriptoren schneller und deutlicher.

SIFT und SURF haben nahezu den gleichen Verlauf und schneiden am besten ab. BRIEF und ORB liefern auch nahezu die gleichen Werte und sind fast so gut wie SIFT und SURF. BRISK und FREAK leisten hier am wenigsten. BRSIK ist aber noch besser als FREAK.

Bei dem Set der Bäume wurde das Kameraobjektv so verstellt, das der Großteil der späteren Bilder unscharf ist.

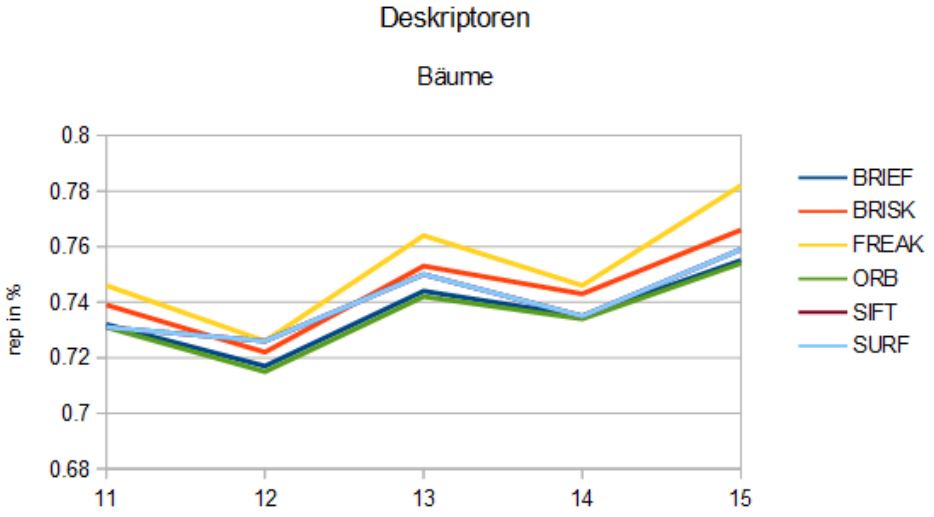


Abbildung 4.30.: Bäume

Hier schneiden alle Deskriptoren in etwa gleichgut ab. Nur FREAK etwas besser als die anderen. Das Interessante ist, dass das letzte Bild sich besser mit dem Ersten Matchen lässt, als das erste Bild mit sich selbst. Das stimmt erstaunlicherweise. Es liegt daran, dass das erste Bild zu detailliert ist (hier sind ist der Hauptbereich des Bildes noch gestochen scharf), das in den direkten Umgebungen der Merkmale die Pixel schon zu unterschiedlich

sind, als das eine perfekte Beschreibung möglich wäre.

Bei den Bilderserien Graffiti und Mauer wurde der Gesichtspunkt verändert (wie schon oben erklärt).

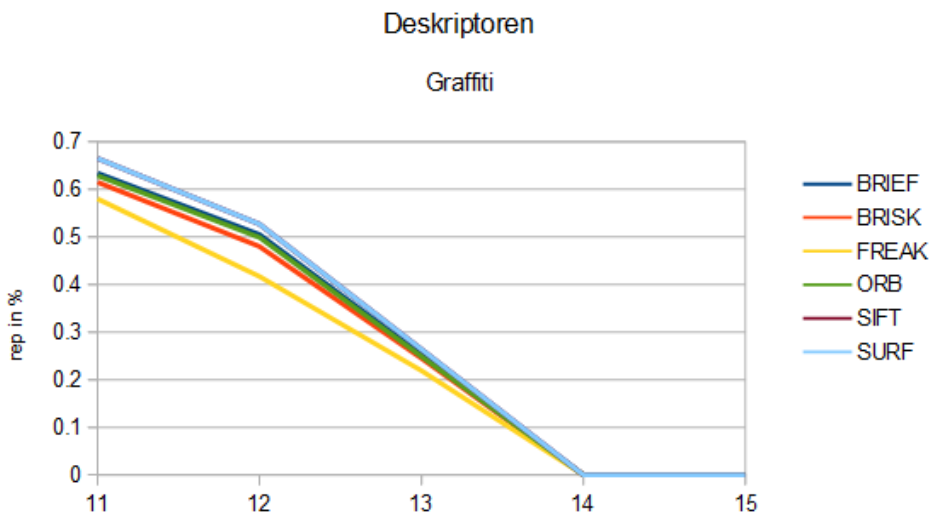


Abbildung 4.31.: Graffiti

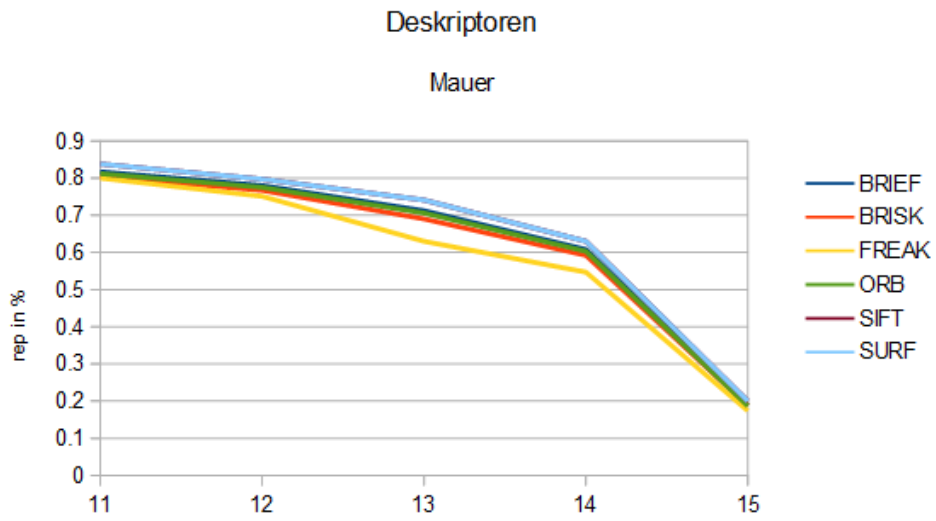


Abbildung 4.32.: Mauer

Hier verhält es sich ähnlich wie bei den Detektoren. Alles kommen fast gleich gut mit den Umständen zurecht. FREAK ist etwas weniger gut als die anderen. Bei den letzten beiden Bilder der Graffitiserie liefert gar kein Deskriptor mehr ein Ergebnis.

Zur JPEG Kompression lässt sich festhalten, dass alle Deskriptoren gleichgut damit zurechtkommen:

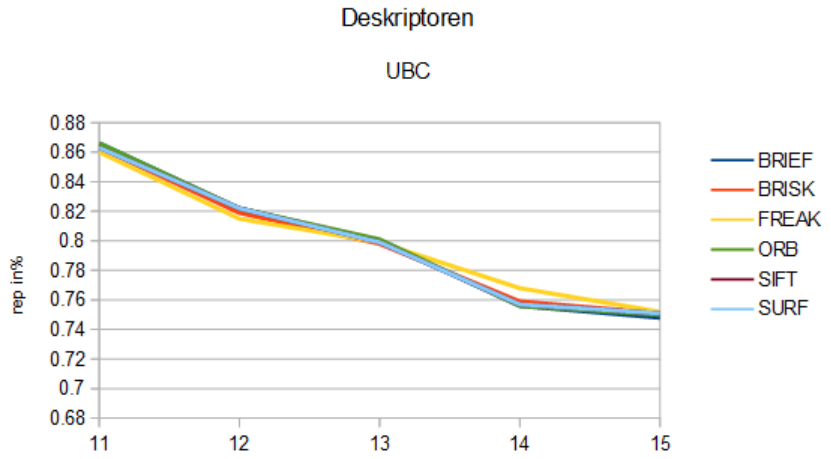


Abbildung 4.33.: UBC

Zeit

Als nächstes ist das Diagramm zu Deskriptionszeit zu sehen:

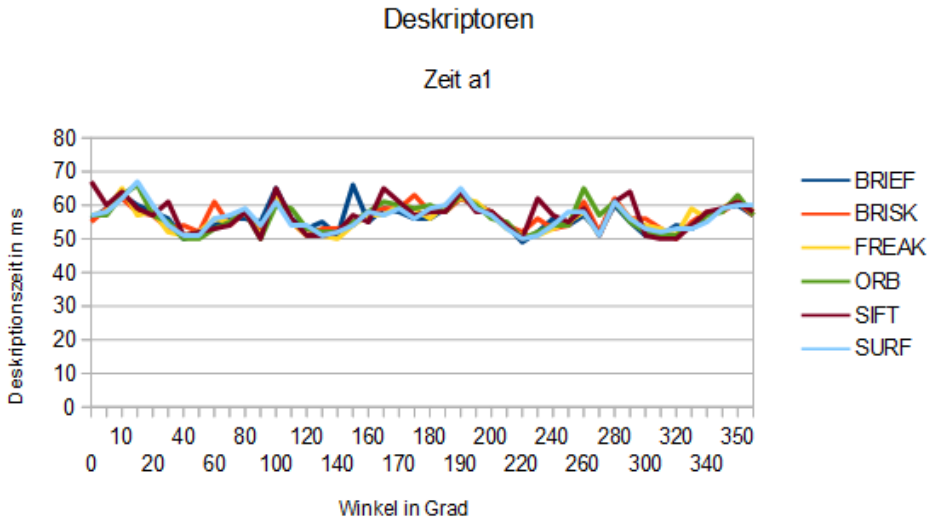


Abbildung 4.34.: Zeit

Im Gegensatz zu den Detektoren gibt es hier nicht so eindeutige Unterschiede zwischen den verschiedenen Algorithmen. Maximalst liegt hier ein Unterschied von 10% mehr oder weniger zu einem zum anderen Deskriptor vor.

Für die Deskriptoren ist es deutlich schwerer Aussagen abzugeben, welcher deutlich genauer oder schneller ist als die Anderen. ORB hat etwas die Nase vorn, SIFT und FREAK schneiden leicht schlechter als der Rest ab. Um den passenden Deskriptor zu finden sollten am besten Tests mit Beispieldaten, die ähnlich zu den Daten sind, die am Ende verarbeitet werden sollen.

5. Implementierung (Aufbau des Programmcodes)

In diesem Kapitel wird der Progammcode erläutert, mit dem die Ergebissedaten erzeugt wurden. Der Source Code selbst liegt auf der DVD der Arbeit bei und wird deshalb hier nicht wirklich zitiert.

Als Inspiration dienten die Codingbeispiele aus [28] und [27]. Sowie die OpenCV Beispiele. Und als Lektüre zu OpenCV stellenweise [31]. Damit die Software fehlerfrei kompiliert sollte die 2.4.5er Version von OpenCV verwendet werden.

Zuerst werden ganz kurz drei kleine Funktionen erklärt:

Die Funktion *currentDateTime()* gibt die aktuelle Zeit aus. Sie wurde verwendet um einen Nullpunkt für die Zeit zu definieren. Somit kann die Zeit relativ berechnet werden.

Die Funktion *euclideanDist(Point2f& p, Point2f& q)* gibt den euklidischen Abstand zwischen zwei Punkten im Format *Point2f* aus.

Eine Funktion, welche im finalen Programm nicht mehr benutzt wird, gibt eine Matrix in der Kommandozeile übersichtlich mit einer zu übergebenden Präzision aus. Sie lautet: *print(Mat mat, int prec)*.

Bevor auf die entscheidenden Funktionen, welche die Detektoren und Deskriptoren auf die Bilddaten anwenden und die Ergebnisdaten produzieren, wird jetzt kurz die Hauptfunktion beschrieben.

Die Methode *int main()* besteht aus mehreren verschachtelten Schleifen. Sie durchläuft alle Detektor- und Deskriptorkombinationen für alle Bilddaten (Einzelbilder und Serien).

Nur die Kombination von SIFT als Detektor und ORB als Deskriptor wird nicht durchgerechnet, weil sie nicht funktioniert. Theoretisch müssten die beiden Algorithmen zusammenarbeiten können, aber in der Software traten dabei nicht trivial zu überwindende Hürden auf.

Die Main-Methode ruft die folgenden Funktionen:

```
1  head = calcsinglerot(detectors[i].c_str(), descriptors[j].c_str(), ↩
       imgs[k].c_str(), true, head); // rotation
2  calcsinglesca(detectors[i].c_str(), descriptors[j].c_str(), imgs[k].↩
       c_str(), true); // scale
3  calcsinglelig(detectors[i].c_str(), descriptors[j].c_str(), imgs[k].↩
       c_str(), true); // lightening
4  calcsingleblu(detectors[i].c_str(), descriptors[j].c_str(), imgs[k].↩
       c_str(), true); // blur
5  calcset(detectors[i].c_str(), descriptors[j].c_str(), datasets[l].c_str↩
       (), true); // sets
```

auf.

Die Namen der Funktionen stehen für Berechnung (*cal*) von Einzelbildern (*single*) oder Sets (*set*). Bei den Berechnungen der Einzelbilder steht noch die durchgeführte Bildmodifikationen dabei. Sie ist im Zitat dem Progammkommentar zu entnehmen.

Nur die erste Funktion hat einen Rückgabewert. Sie sollte als erste aufgerufen werden, weil hier die Kopfzeile für die Datanausgabe erstellt wird. In 4.1.2 wurde die Kopfzeile erklärt.

Als Übergabewerte benötigen alle Funktionen, einen Detektor und Deskriptor sowie zu verarbeitenden Bilddaten. An vierter Stelle steht immer ein *true*. Das ist eine Flag für *findHom*. Wenn es auf wahr steht, wird versucht eine Homographie zu finden. Zum Testen neuer Programmteile ist es nützlich dies ausschalten zu können, weil dort am ehesten Unstimmigkeiten auftreten können.

Im folgenden wird nur noch auf die erste und letzte Methode eingangen, da die Zweite bis Vierte der Ersten sehr ähnlich sind. Die genauen Unterschiede können dem Source Code entnommen werden.

Nun zur Funktion *calcsinglerot*. In dem Quellcode sieht ihre Kopfzeile folgendermaßen aus:

```
1 bool calcsinglerot(string detector_name, string descriptor_name, string ↩
    imgname, bool findHom, bool phead)
```

Zur Besseren Nachvollziehbarkeit der kurzen Erläuterungen bietet es sich an den Quellcode parallel zu Lesen.

Nachdem Variablen definiert, das Bild eingelesen, die Ausgabedatei geöffent und deren Kopfzeile ausgegeben wurde, werden der entsprechende Detektor, Deskriptor und Matcher (Bruteforce für Hammingdistanz oder euklidischen Abstand (*L2*)) definiert.

Ein Feld mit den Drehwinkeln wird angelegt. (Bei den anderen Funktionen eben ein Feld mit deren passender Parameter, wie z.B. dem Skalierungsfaktor.)

Nun wird das Feld mit einer Schleife, die den restlichen Teil der Methode umschließt, durchlaufen. Das Bild wird entsprechend modifiziert und auch mit dem Detektor und Deskriptor verarbeitet.

Die Merkmalsvektoren der Punkte werden gematcht und aussortiert wenn sie nicht einer bestimmten Güte genügen. Unter dem Punkt **#gmatch** in 4.1.2 ist der Wert von 0.6 erläutert.

Die nächste kleine Schleife formatiert die Matches, zur Weiterverarbeitung, in eine andere Variable um.

Nun werden die Merkmale (wenn welche gefunden wurden) ensprechend des Rotationswinkels tranformiert. Danach kann kontrolliert werden, wie viele Punkte richtig zugeordnet wurden.

Die entscheidenen Werte (**per** und **matrat**) sind damit berechnet.

Als letztes wird noch die Homographiematrix berechnet und die Funktion *evaluateFeatureDetector* aufgerufen. Die Resultate davon (für Einzelbilder) gehen zwar nicht in die Ergebisse dieser Arbeit ein, können aber gerne in aufbauenden Arbeiten benutzt werden. Sie liegen der Arbeit ebenfalls digital bei.

Am Ende werden der Detektor, der Deskriptor und der Matcher gelöscht um den Speicher wieder freizugeben.

Zwischendrin werden die Ergebniswerte in die Datei geschrieben. Wann genau das passiert ist im Quelltext einfach zu erkennen.

Am Ende dieses Kapitels wird noch auf die Methode *calcset* eingegangen. In dem Quellcode sieht ihre Kopfzeile aus wie folgt:

```
1 calcset(string detector_name, string descriptor_name, string datasetname, ↩
    bool findHom)
```

Da die Funktion viele Gemeinsamkeiten mit der eben Erklärten hat wird jetzt der Fokus auf die entscheidenden Unterschiede (zwischen den Methoden) gelegt.

Statt einem Bild wird hier die ganze Bilderserie (immer sechs Bilder) eingelesen.

Statt die Parameter (wie z.B. Rotationswinkel) zu durchlaufen, werden jetzt die Bilder der Reihe nach mit dem ersten Bild verglichen. Zur Kontrolle auch das Erste mit sich selbst.

Die verbleibenden Unterschiede sind, dass die Werte **per** und **matrat** nicht berechnet werden und die Homographiematrix nicht berrechnet sondern auch eingelesen wird. Sie wird mit den Bildserien zusammen zur Verfügung gestellt.

Der Rest geschied analog wie in *calcsinglerot*.

6. Konklusion

Am Ende der Unterkapitel 4.2.1 und 4.2.2 wurden schon Schlüsse gezogen, welcher Detektoren und Deskriptoren unter bestimmten Umständen am besten verwendet werden sollten.

Die Quellen in 3 liefern nahezu die gleichen Ergebisse, wie die im Rahmen dieser Arbeit durchgeführten Berechnungen. Auch die Diagramme in [26] und [27] sehen den in 4 erarbeiteten Diagrammen sehr ähnlich.

Die Resultate entsprechen den Erwartungen. Der Mehrwert dieser Arbeit ist aber der umfangreichere Vergleich von Merkmalserkennungs und -beschreibungsalgorithmen, als in allen anderen vorliegenden Quellen. Dazu sind mehr Daten produziert worden, als ausgewertet werden konnten. Mehr dazu im nächsten Kapitel.

7. Ausblick

Eine aufbauende Arbeit zu der vorliegenden Abhandlung könnte die Daten, welche die Software ausgegeben hat noch weiter analysieren und aufschlüsseln. Es wurde für jede Detektor-Deskriptor-Kombination (außer einer) eine Ergebnissdatei erstellt. Dass sind 41 Dateien, von denen aber letztlich nur 13 ausgewertet werden konnten (um auch die anderen Kapitel nicht zu vernachlässigen). Selbst aus diesen Dateien wurde nur ein Teil der Daten, zum Erstellen der Ergebnissdiagramme verwendet.

Zusätzlich sind noch vier weitere Richtungen für aufbauende Arbeiten möglich. Da der Merkmalswiederekennungsprozess aus drei Schritten besteht kann an jedem einzelnen Teil angesetzt werden.

Zum ersten könnte die Software genutzt werden um noch weitere Detektoren in den Vergleich miteinzubeziehen. In OpenCV sollten inzwischen auch die Detektoren KAZE ([32]) bzw. AKAZE implementiert sein. Auch auf einen AGAST Corner Detector wurde während der Recherche gestoßen. Die Abkürzung steht für Adaptive and Generic Corner Detection Based on the Accelerated Segment Test.

Ebenfalls besteht die Möglichkeit weitere Deskriptoren mit in den Vergleich aufzunehmen. Als Beispiel dient hier der HoG-Deskriptor ([33]).

Der Schritt (der Feature detection) an dem noch am meisten Performance heraus geholt werden kann, ist der dritte und letzte Schritt: Das Matching. In dieser Abhandlung wurden nur ein Matchingalgorithmus betrachtet, weil der Fokus auf den anderen Schritten lag. Mit z.B. FLANN ([17]) lässt sich der ganze Prozess sicher noch sehr beschleunigen.

Als letzter Punkt können natürlich (auch recht einfach) weitere Bilder und Bilderserien mit der Software analysiert werden. Das bedeutet vermutlich den geringsten Programmieraufwand und liefert aber höchstwahrscheinlich auch aussagekräftige Resultate.

Literaturverzeichnis

[1] "Opencv-python tutorials » feature detection and description » understanding features." http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," 2004.

[3] P. J. Burt and E. H. Adelson, "The laplacian pyramid as a compact image code," *IEEE TRANSACTIONS ON COMMUNICATIONS*, vol. 31, pp. 532–540, 1983.

[4] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *In ECCV*, pp. 404–417, 2006.

[5] "Opencv 3.0.0-dev documentation » opencv-python tutorials » feature detection and description » introduction to surf (speeded-up robust features)." http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html.

[6] Y. Biadgie and K.-A. Sohn, "Feature detector using adaptive accelerated segment test,"

[7] J. Shi and C. Tomasi, "Good features to track," 1994.

[8] "Opencv 3.0.0-dev documentation » opencv-python tutorials » feature detection and description » harris corner detection." http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners.

[9] "Opencv 3.0.0-dev documentation » opencv-python tutorials » feature detection and description » shi-tomasi corner detector & good features to track." http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html{#}shi-tomasi.

[10] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in *In British Machine Vision Conference*, pp. 384–393, 2002.

[11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *In ICCV*.

[12] M. Agrawal, K. Konolige, and M. R. Blas, "Censure: Center surround extremas for realtime feature detection and matching," in *Computer Vision ECCV 2008, volume 5305 of Lecture Notes in Computer Science*, pp. 102–115, Springer, 2008.

[13] "Star feature detector." http://experienceopencv.blogspot.de/2011/01/star-feature-detector.html.

[14] M. Calonder, V. Lepetit, M. Özuysal, T. Trzcinski, C. Strecha, and P. Fua, "Brief: Computing a local binary descriptor very fast."

[15] S. Leutenegger, M. Chli, and Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *In Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2548–2555, 2011.

[16] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint,"

[17] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pp. 331–340, INSTICC Press, 2009.

[18] O. Miksik and K. Mikolajczyk, "Evaluation of local detectors and descriptors for fast feature matching."

[19] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, pp. 1615–1630, Oct. 2005.

[20] "University of oxford: Datasets." http://www.apress.com/9781430260790?gtmf=s.

[21] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: A survey," *FnT Comp. Graphics and Vision*, pp. 177–280, 2008.

[22] S. Gauglitz, T. Höllerer, and M. Turk, "Evaluation of interest point detectors and feature descriptors for visual tracking," *Int. J. Comput. Vision*, vol. 94, pp. 335–360, Sept. 2011.

[23] P. Moreels and P. Perona, "Evaluation of features detectors and descriptors based on 3d objects," in *IJCV*, pp. 800–807, 2005.

[24] "Computer vision talks." http://computer-vision-talks.com/.

[25] "Comparison of the opencv's feature detection algorithms." http://computer-vision-talks.com/articles/2011-01-04-comparison-of-the-opencv-feature-detection-algorithms/.

[26] "Feature descriptor comparison report." http://computer-vision-talks.com/articles/2011-08-19-feature-descriptor-comparison-report/.

[27] "A battle of three descriptors: Surf, freak and brisk." http://computer-vision-talks.com/articles/2012-08-18-a-battle-of-three-descriptors-surf-freak-and-brisk/.

[28] S. Brahmbhatt, *Practical OpenCV*. New York: Apress, 2013. aufl. ed., 2013.

[29] "Practical opencv: Source code/downloads." http://www.robots.ox.ac.uk/~vgg/data/data-aff.html.

[30] "Opencv docs: evaluatefeaturedetector." http://docs.opencv.org/3.0.0/da/d9b/group__features2d.html#gac78532caa0ee7744ab370c9a479ec397.

[31] G. R. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. Sebastopol and CA: O'Reilly, 1 ed., 2008.

[32] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "Kaze features."

[33] "Hog descriptor." http://experienceopencv.blogspot.de/2011/02/hog-descriptor.html.

A. Appendix

A.1. Inhalt der DVD

Der Inhalt der beiliegenden DVD ist wie folgt aufgebaut:

- 0_Quellen
- 1_Daten
- 2_Software
- 3_Latex
- 4_Riegler_MA.pdf

Unter Quellen sind alle Referzen zu finden die als .pdf im Internet verfügbar waren.

In Daten sind die Eingangsdaten (Bilder) und Ergebisse (Softwareausgabe) zu finden.

In Software liegt die main.cpp, des Progamms, die passenden Bilder sind 1_Daten zu entnehmen.

In Latex liegt erst diese Masterarbeit als .tex Datei. Und ein Ordner mit allem das zum kompilieren notwenig ist. Der ist aber aufgrund der Menge an Dateien nicht sonderlich übersichtlich.

Als letzte liegt diese Masterarbeit als .pdf auf der DVD.

A.2. Weitere Bilder, die verwendet wurden

b1 b2 b3 b4

Abbildung A.1.: Verwendete Bilder (2)

Boot 1 Boot 2 Boot 3 Boot 4 Boot 5 Boot 6

Abbildung A.2.: Set Boot

Graffiti 1 Graffiti 2 Graffiti 3 Graffiti 4 Graffiti 5 Graffiti 6

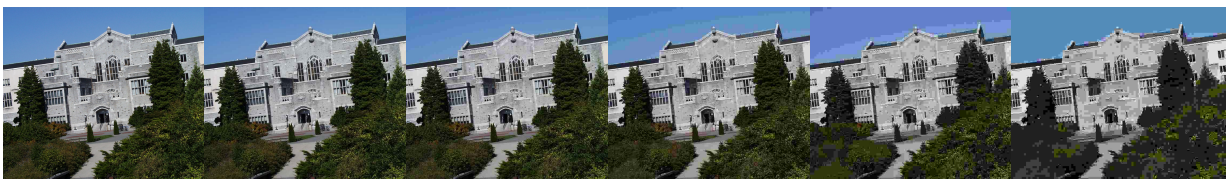
Abbildung A.3.: Set Graffiti

Leuven 1 Leuven 2 Leuven 3 Leuven 4 Leuven 5 Leuven 6

Abbildung A.4.: Set Leuven

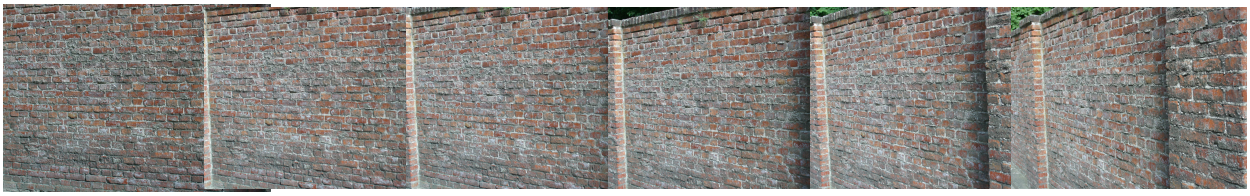
Bäume 1 Bäume 2 Bäume 3 Bäume 4 Bäume 5 Bäume 6

Abbildung A.5.: Set Bäume



UBC 1 UBC 2 UBC 3 UBC 4 UBC 5 UBC 6

Abbildung A.6.: Set UBC



Mauer 1 Mauer 2 Mauer 3 Mauer 4 Mauer 5 Mauer 6

Abbildung A.7.: Set Mauer

A.3. Weitere Diagramme zum Rotationstest der Detektoren

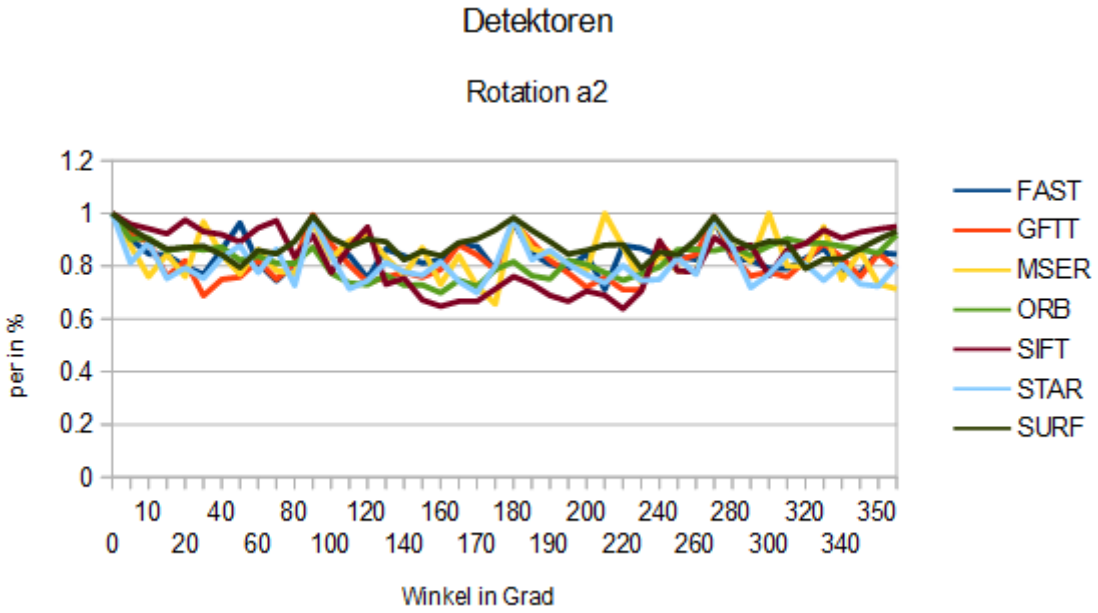


Abbildung A.8.: Detektoren Rotation a2

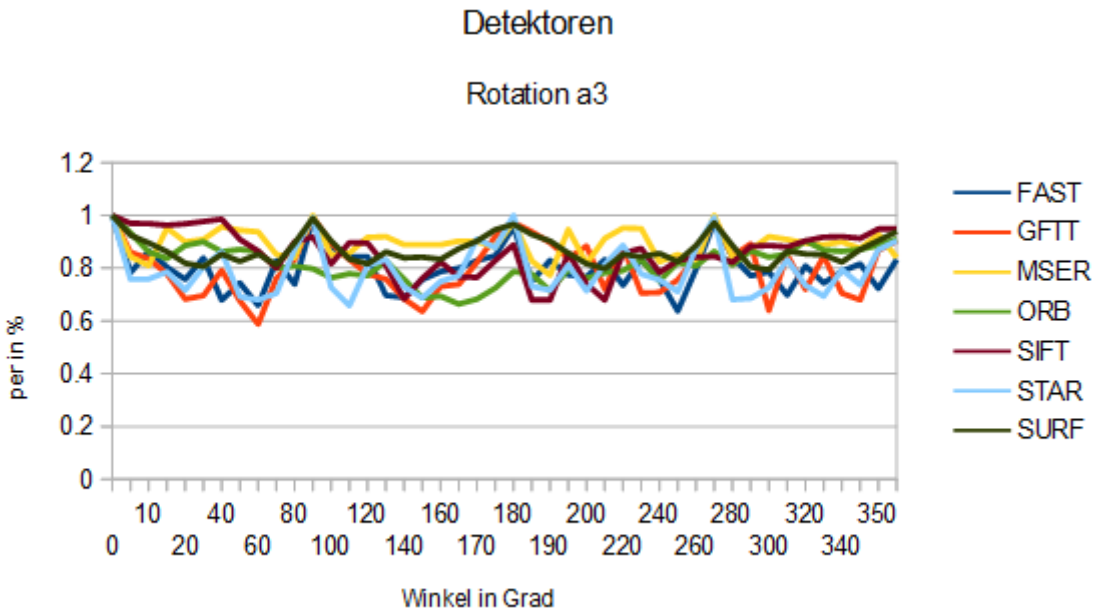


Abbildung A.9.: Detektoren Rotation a3

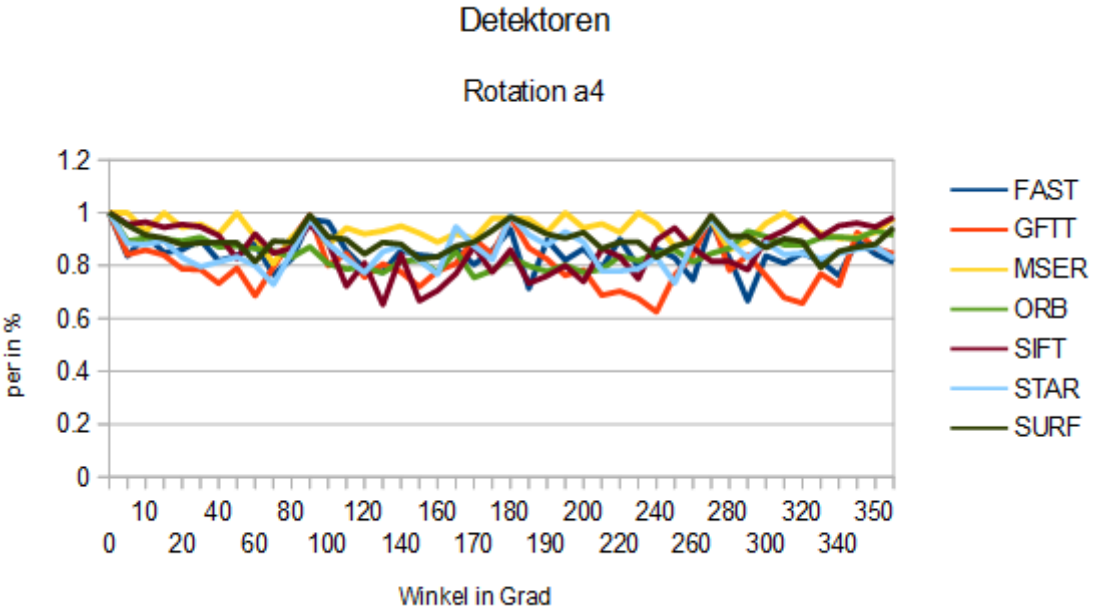


Abbildung A.10.: Detektoren Rotation a4

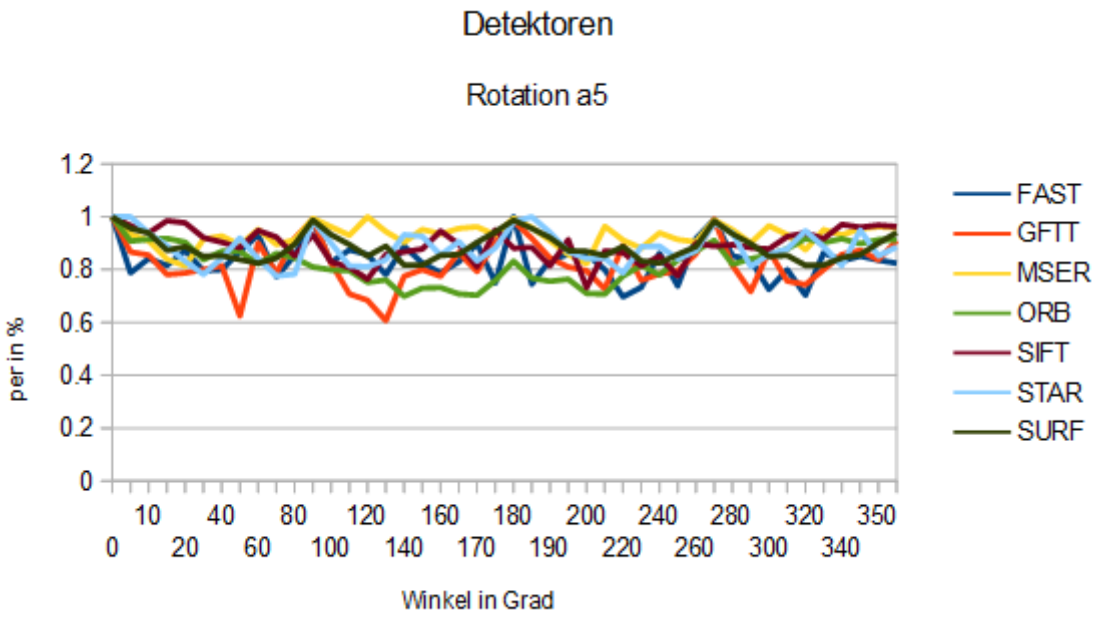


Abbildung A.11.: Detektoren Rotation a5

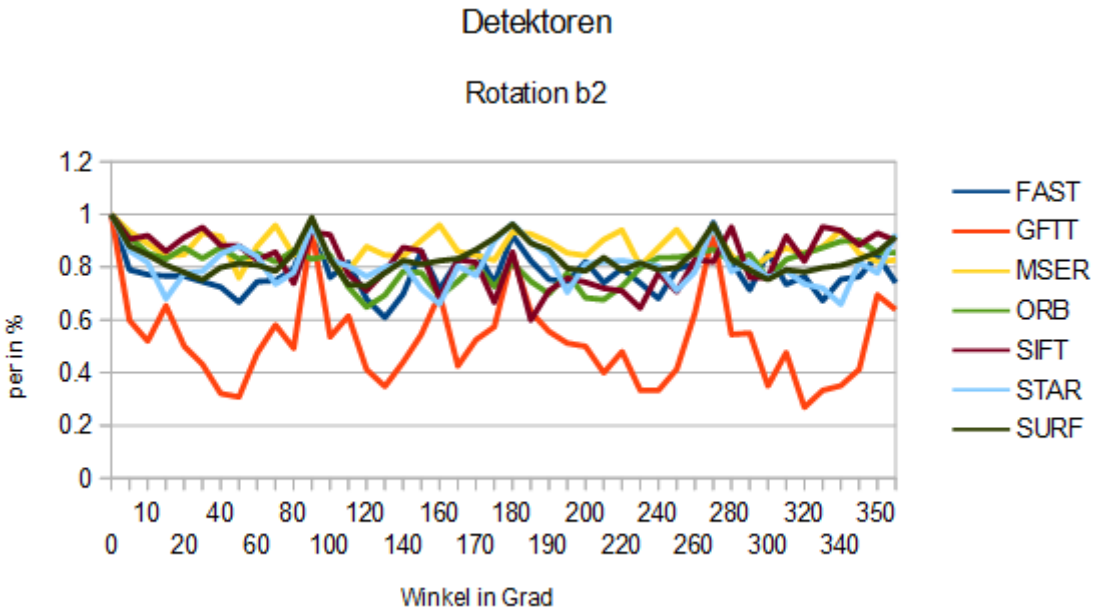


Abbildung A.12.: Detektoren Rotation b2

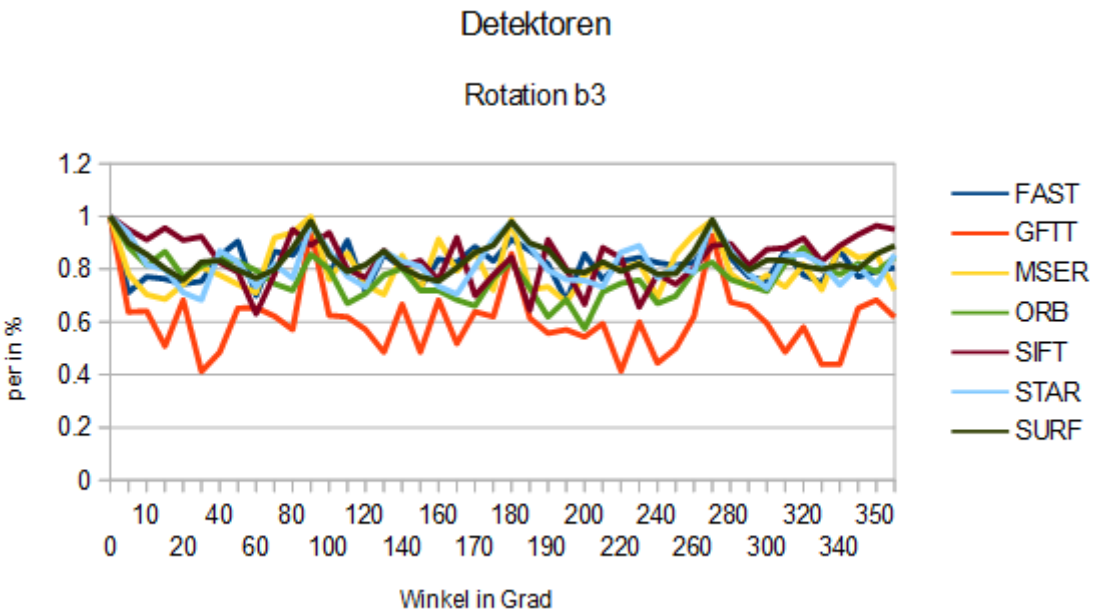


Abbildung A.13.: Detektoren Rotation b3